

---

## Description of STM32F1xx 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 STM32CubeF1 for STM32F1 series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
  - All embedded software utilities coming with a full set of examples.

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

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

The HAL drivers APIs are split into two categories: generic APIs which provide common and generic functions for all the STM32 series and extension APIs which include specific and customized functions for a given family or part number.

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

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

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

This user manual is structured as follows:

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



## Contents

<b>1</b>	<b>Acronyms and definitions.....</b>	<b>36</b>
<b>2</b>	<b>Overview of HAL drivers.....</b>	<b>38</b>
2.1	HAL and user-application files.....	38
2.1.1	HAL driver files .....	38
2.1.2	User-application files .....	39
2.2	HAL data structures .....	41
2.2.1	Peripheral handle structures .....	41
2.2.2	Initialization and configuration structure .....	42
2.2.3	Specific process structures .....	43
2.3	API classification .....	43
2.4	Devices supported by HAL drivers .....	44
2.5	HAL drivers rules.....	51
2.5.1	HAL API naming rules .....	51
2.5.2	HAL general naming rules.....	52
2.5.3	HAL interrupt handler and callback functions.....	53
2.6	HAL generic APIs.....	54
2.7	HAL extension APIs .....	55
2.7.1	HAL extension model overview .....	55
2.7.2	HAL extension model cases .....	55
2.8	File inclusion model.....	57
2.9	HAL common resources.....	58
2.10	HAL configuration.....	59
2.11	HAL system peripheral handling .....	60
2.11.1	Clock.....	60
2.11.2	GPIOs.....	60
2.11.3	Cortex NVIC and SysTick timer.....	62
2.11.4	PWR .....	62
2.11.5	EXTI.....	62
2.11.6	DMA.....	64
2.12	How to use HAL drivers .....	66
2.12.1	HAL usage models .....	66
2.12.2	HAL initialization .....	67
2.12.3	HAL IO operation process .....	69
2.12.4	Timeout and error management.....	72

<b>3</b>	<b>HAL System Driver .....</b>	<b>76</b>
3.1	HAL Firmware driver API description .....	76
3.1.1	How to use this driver .....	76
3.1.2	Initialization and de-initialization functions .....	76
3.1.3	HAL Control functions.....	76
3.1.4	HAL_Init.....	77
3.1.5	HAL_DeInit .....	77
3.1.6	HAL_MspInit .....	77
3.1.7	HAL_MspDeInit .....	78
3.1.8	HAL_InitTick .....	78
3.1.9	HAL_IncTick .....	78
3.1.10	HAL_GetTick .....	78
3.1.11	HAL_Delay .....	78
3.1.12	HAL_SuspendTick.....	78
3.1.13	HAL_ResumeTick.....	79
3.1.14	HAL_GetHalVersion .....	79
3.1.15	HAL_GetREVID .....	79
3.1.16	HAL_GetDEVID.....	79
3.1.17	HAL_DBGMCU_EnableDBGSleepMode .....	79
3.1.18	HAL_DBGMCU_DisableDBGSleepMode .....	79
3.1.19	HAL_DBGMCU_EnableDBGStopMode .....	80
3.1.20	HAL_DBGMCU_DisableDBGStopMode .....	80
3.1.21	HAL_DBGMCU_EnableDBGStandbyMode .....	80
3.1.22	HAL_DBGMCU_DisableDBGStandbyMode .....	80
3.2	HAL Firmware driver defines.....	81
3.2.1	HAL.....	81
<b>4</b>	<b>HAL ADC Generic Driver.....</b>	<b>82</b>
4.1	ADC Firmware driver registers structures .....	82
4.1.1	ADC_InitTypeDef.....	82
4.1.2	ADC_ChannelConfTypeDef .....	83
4.1.3	ADC_AnalogWDGConfTypeDef.....	84
4.1.4	ADC_HandleTypeDef .....	84
4.2	ADC Firmware driver API description.....	85
4.2.1	ADC peripheral features .....	85
4.2.2	How to use this driver .....	85
4.2.3	Initialization and de-initialization functions .....	88
4.2.4	IO operation functions .....	88

4.2.5	Peripheral Control functions .....	89
4.2.6	Peripheral State and Errors functions .....	89
4.2.7	HAL_ADC_Init .....	89
4.2.8	HAL_ADC_DeInit .....	89
4.2.9	HAL_ADC_MspInit .....	90
4.2.10	HAL_ADC_MspDeInit .....	90
4.2.11	HAL_ADC_Start .....	90
4.2.12	HAL_ADC_Stop .....	90
4.2.13	HAL_ADC_PollForConversion .....	91
4.2.14	HAL_ADC_PollForEvent .....	91
4.2.15	HAL_ADC_Start_IT .....	91
4.2.16	HAL_ADC_Stop_IT .....	91
4.2.17	HAL_ADC_Start_DMA .....	91
4.2.18	HAL_ADC_Stop_DMA .....	92
4.2.19	HAL_ADC_GetValue .....	92
4.2.20	HAL_ADC_IRQHandler .....	92
4.2.21	HAL_ADC_ConvCpltCallback .....	92
4.2.22	HAL_ADC_ConvHalfCpltCallback .....	93
4.2.23	HAL_ADC_LevelOutOfWindowCallback .....	93
4.2.24	HAL_ADC_ErrorCallback .....	93
4.2.25	HAL_ADC_ConfigChannel .....	93
4.2.26	HAL_ADC_AnalogWDGConfig .....	94
4.2.27	HAL_ADC_GetState .....	94
4.2.28	HAL_ADC_GetError .....	94
4.3	ADC Firmware driver defines .....	94
4.3.1	ADC .....	94
<b>5</b>	<b>HAL ADC Extension Driver .....</b>	<b>106</b>
5.1	ADCEX Firmware driver registers structures .....	106
5.1.1	ADC_InjectionConfTypeDef .....	106
5.1.2	ADC_MultiModeTypeDef .....	107
5.2	ADCEX Firmware driver API description .....	108
5.2.1	IO operation functions .....	108
5.2.2	Peripheral Control functions .....	109
5.2.3	HAL_ADCEX_Calibration_Start .....	109
5.2.4	HAL_ADCEX_InjectedStart .....	109
5.2.5	HAL_ADCEX_InjectedStop .....	109
5.2.6	HAL_ADCEX_InjectedPollForConversion .....	109
5.2.7	HAL_ADCEX_InjectedStart_IT .....	110

5.2.8	HAL_ADCEx_InjectedStop_IT .....	110
5.2.9	HAL_ADCEx_MultiModeStart_DMA .....	110
5.2.10	HAL_ADCEx_MultiModeStop_DMA.....	110
5.2.11	HAL_ADCEx_InjectedGetValue .....	111
5.2.12	HAL_ADCEx_MultiModeGetValue .....	111
5.2.13	HAL_ADCEx_InjectedConvCpltCallback .....	111
5.2.14	HAL_ADCEx_InjectedConfigChannel .....	111
5.2.15	HAL_ADCEx_MultiModeConfigChannel .....	112
5.3	ADCEx Firmware driver defines .....	112
5.3.1	ADCEx.....	112
<b>6</b>	<b>HAL CAN Generic Driver.....</b>	<b>118</b>
6.1	CAN Firmware driver registers structures .....	118
6.1.1	CAN_InitTypeDef.....	118
6.1.2	CanTxMsgTypeDef.....	118
6.1.3	CanRxMsgTypeDef .....	119
6.1.4	CAN_HandleTypeDef .....	120
6.2	CAN Firmware driver API description.....	120
6.2.1	How to use this driver .....	120
6.2.2	Initialization and de-initialization functions .....	121
6.2.3	IO operation functions .....	122
6.2.4	Peripheral State and Error functions .....	122
6.2.5	HAL_CAN_Init .....	122
6.2.6	HAL_CAN_ConfigFilter.....	122
6.2.7	HAL_CAN_DeInit.....	123
6.2.8	HAL_CAN_MspInit .....	123
6.2.9	HAL_CAN_MspDeInit.....	123
6.2.10	HAL_CAN_Transmit.....	123
6.2.11	HAL_CAN_Transmit_IT.....	123
6.2.12	HAL_CAN_Receive .....	124
6.2.13	HAL_CAN_Receive_IT.....	124
6.2.14	HAL_CAN_Sleep.....	124
6.2.15	HAL_CAN_WakeUp .....	124
6.2.16	HAL_CAN_IRQHandler .....	125
6.2.17	HAL_CAN_TxCpltCallback.....	125
6.2.18	HAL_CAN_RxCpltCallback .....	125
6.2.19	HAL_CAN_ErrorCallback .....	125
6.2.20	HAL_CAN_GetState.....	126
6.2.21	HAL_CAN_GetError .....	126

6.3	CAN Firmware driver defines .....	126
6.3.1	CAN .....	126
<b>7</b>	<b>HAL CAN Extension Driver .....</b>	<b>136</b>
7.1	CANEx Firmware driver registers structures .....	136
7.1.1	CAN_FilterConfTypeDef .....	136
7.2	CANEx Firmware driver defines .....	137
7.2.1	CANEx .....	137
<b>8</b>	<b>HAL CEC Generic Driver .....</b>	<b>138</b>
8.1	CEC Firmware driver registers structures .....	138
8.1.1	CEC_InitTypeDef .....	138
8.1.2	CEC_HandleTypeDef .....	138
8.2	CEC Firmware driver API description .....	139
8.2.1	How to use this driver .....	139
8.2.2	Initialization and Configuration functions .....	139
8.2.3	IO operation functions .....	139
8.2.4	Peripheral Control functions .....	140
8.2.5	HAL_CEC_Init .....	140
8.2.6	HAL_CEC_DeInit .....	140
8.2.7	HAL_CEC_MspInit .....	141
8.2.8	HAL_CEC_MspDeInit .....	141
8.2.9	HAL_CEC_Transmit .....	141
8.2.10	HAL_CEC_Receive .....	141
8.2.11	HAL_CEC_Transmit_IT .....	142
8.2.12	HAL_CEC_Receive_IT .....	142
8.2.13	HAL_CEC_GetReceivedFrameSize .....	142
8.2.14	HAL_CEC_IRQHandler .....	143
8.2.15	HAL_CEC_TxCpltCallback .....	143
8.2.16	HAL_CEC_RxCpltCallback .....	143
8.2.17	HAL_CEC_ErrorCallback .....	143
8.2.18	HAL_CEC_GetState .....	143
8.2.19	HAL_CEC_GetError .....	143
8.3	CEC Firmware driver defines .....	144
8.3.1	CEC .....	144
<b>9</b>	<b>HAL CORTEX Generic Driver .....</b>	<b>150</b>
9.1	CORTEX Firmware driver API description .....	150
9.1.1	Initialization and de-initialization functions .....	150
9.1.2	Peripheral Control functions .....	150

9.1.3	HAL_NVIC_SetPriorityGrouping .....	150
9.1.4	HAL_NVIC_SetPriority .....	151
9.1.5	HAL_NVIC_EnableIRQ .....	151
9.1.6	HAL_NVIC_DisableIRQ.....	151
9.1.7	HAL_NVIC_SystemReset.....	152
9.1.8	HAL_SYSTICK_Config.....	152
9.1.9	HAL_NVIC_GetPriorityGrouping .....	152
9.1.10	HAL_NVIC_GetPriority .....	152
9.1.11	HAL_NVIC_SetPendingIRQ .....	153
9.1.12	HAL_NVIC_GetPendingIRQ .....	153
9.1.13	HAL_NVIC_ClearPendingIRQ.....	153
9.1.14	HAL_NVIC_GetActive .....	153
9.1.15	HAL_SYSTICK_CLKSourceConfig .....	154
9.1.16	HAL_SYSTICK_IRQHandler .....	154
9.1.17	HAL_SYSTICK_Callback .....	154
9.2	CORTEX Firmware driver defines.....	154
9.2.1	CORTEX.....	154
<b>10</b>	<b>HAL CRC Generic Driver.....</b>	<b>156</b>
10.1	CRC Firmware driver registers structures .....	156
10.1.1	CRC_HandleTypeDef.....	156
10.2	CRC Firmware driver API description .....	156
10.2.1	How to use this driver .....	156
10.2.2	Initialization and de-initialization functions .....	156
10.2.3	Peripheral Control functions .....	157
10.2.4	Peripheral State functions .....	157
10.2.5	HAL_CRC_Init.....	157
10.2.6	HAL_CRC_DeInit .....	157
10.2.7	HAL_CRC_MspInit .....	157
10.2.8	HAL_CRC_MspDeInit.....	158
10.2.9	HAL_CRC_Accumulate .....	158
10.2.10	HAL_CRC_Calculate.....	158
10.2.11	HAL_CRC_GetState.....	158
10.3	CRC Firmware driver defines .....	159
10.3.1	CRC.....	159
<b>11</b>	<b>HAL DAC Generic Driver.....</b>	<b>160</b>
11.1	DAC Firmware driver registers structures .....	160
11.1.1	DAC_HandleTypeDef .....	160

11.1.2	DAC_ChannelConfTypeDef .....	160
11.2	DAC Firmware driver API description .....	161
11.2.1	DAC Peripheral features .....	161
11.2.2	How to use this driver .....	162
11.2.3	Initialization and de-initialization functions .....	163
11.2.4	IO operation functions .....	164
11.2.5	Peripheral Control functions .....	164
11.2.6	Peripheral State and Errors functions .....	164
11.2.7	HAL_DAC_Init .....	164
11.2.8	HAL_DAC_DeInit .....	165
11.2.9	HAL_DAC_MspInit .....	165
11.2.10	HAL_DAC_MspDeInit .....	165
11.2.11	HAL_DAC_Start .....	165
11.2.12	HAL_DAC_Stop .....	166
11.2.13	HAL_DAC_Start_DMA .....	166
11.2.14	HAL_DAC_Stop_DMA .....	166
11.2.15	HAL_DAC_GetValue .....	167
11.2.16	HAL_DAC_ConvCpltCallbackCh1 .....	167
11.2.17	HAL_DAC_ConvHalfCpltCallbackCh1 .....	167
11.2.18	HAL_DAC_ErrorCallbackCh1 .....	167
11.2.19	HAL_DAC_SetValue .....	168
11.2.20	HAL_DAC_ConfigChannel .....	168
11.2.21	HAL_DAC_SetValue .....	168
11.2.22	HAL_DAC_GetState .....	169
11.2.23	HAL_DAC_GetError .....	169
11.2.24	HAL_DAC_ConvCpltCallbackCh1 .....	169
11.2.25	HAL_DAC_ConvHalfCpltCallbackCh1 .....	169
11.2.26	HAL_DAC_ErrorCallbackCh1 .....	170
11.3	DAC Firmware driver defines .....	170
11.3.1	DAC .....	170
<b>12</b>	<b>HAL DAC Extension Driver .....</b>	<b>172</b>
12.1	DACEx Firmware driver API description .....	172
12.1.1	How to use this driver .....	172
12.1.2	Extended features functions .....	172
12.1.3	HAL_DACEx_DualGetValue .....	172
12.1.4	HAL_DACEx_TriangleWaveGenerate .....	172
12.1.5	HAL_DACEx_NoiseWaveGenerate .....	173
12.1.6	HAL_DACEx_DualSetValue .....	174



12.1.7	HAL_DACEx_ConvCpltCallbackCh2 .....	174
12.1.8	HAL_DACEx_ConvHalfCpltCallbackCh2 .....	175
12.1.9	HAL_DACEx_ErrorCallbackCh2 .....	175
12.2	DACEx Firmware driver defines .....	175
12.2.1	DACEx.....	175
<b>13</b>	<b>HAL DMA Generic Driver .....</b>	<b>178</b>
13.1	DMA Firmware driver registers structures .....	178
13.1.1	DMA_InitTypeDef .....	178
13.1.2	__DMA_HandleTypeDef.....	178
13.2	DMA Firmware driver API description .....	179
13.2.1	How to use this driver .....	179
13.2.2	Initialization and de-initialization functions .....	180
13.2.3	IO operation functions .....	180
13.2.4	State and Errors functions .....	181
13.2.5	HAL_DMA_Init.....	181
13.2.6	HAL_DMA_DeInit .....	181
13.2.7	HAL_DMA_Start .....	181
13.2.8	HAL_DMA_Start_IT .....	182
13.2.9	HAL_DMA_Abort .....	182
13.2.10	HAL_DMA_PollForTransfer.....	182
13.2.11	HAL_DMA_IRQHandler.....	183
13.2.12	HAL_DMA_GetState .....	183
13.2.13	HAL_DMA_GetError.....	183
13.3	DMA Firmware driver defines.....	183
13.3.1	DMA.....	183
<b>14</b>	<b>HAL DMA Extension Driver.....</b>	<b>188</b>
14.1	DMAEx Firmware driver defines.....	188
14.1.1	DMAEx.....	188
<b>15</b>	<b>HAL ETH Generic Driver .....</b>	<b>190</b>
15.1	ETH Firmware driver registers structures.....	190
15.1.1	ETH_InitTypeDef .....	190
15.1.2	ETH_MACInitTypeDef .....	190
15.1.3	ETH_DMAInitTypeDef .....	193
15.1.4	ETH_DMADescTypeDef.....	194
15.1.5	ETH_DMARxFramInfos.....	194
15.1.6	ETH_HandleTypeDef .....	195
15.2	ETH Firmware driver API description .....	195

15.2.1	How to use this driver .....	195
15.2.2	Initialization and de-initialization functions .....	196
15.2.3	IO operation functions .....	196
15.2.4	Peripheral Control functions .....	197
15.2.5	Peripheral State functions .....	197
15.2.6	HAL_ETH_Init.....	197
15.2.7	HAL_ETH_DeInit .....	197
15.2.8	HAL_ETH_DMATxDescListInit.....	198
15.2.9	HAL_ETH_DMARxDescListInit .....	198
15.2.10	HAL_ETH_MspltInit.....	198
15.2.11	HAL_ETH_MspDeInit .....	198
15.2.12	HAL_ETH_TransmitFrame .....	199
15.2.13	HAL_ETH_GetReceivedFrame .....	199
15.2.14	HAL_ETH_GetReceivedFrame_IT .....	199
15.2.15	HAL_ETH_IRQHandler .....	199
15.2.16	HAL_ETH_TxCpltCallback .....	199
15.2.17	HAL_ETH_RxCpltCallback.....	200
15.2.18	HAL_ETH_ErrorCallback.....	200
15.2.19	HAL_ETH_ReadPHYRegister .....	200
15.2.20	HAL_ETH_WritePHYRegister .....	200
15.2.21	HAL_ETH_Start.....	201
15.2.22	HAL_ETH_Stop .....	201
15.2.23	HAL_ETH_ConfigMAC .....	201
15.2.24	HAL_ETH_ConfigDMA .....	201
15.2.25	HAL_ETH_GetState .....	202
15.3	ETH Firmware driver defines.....	202
15.3.1	ETH.....	202
<b>16</b>	<b>HAL FLASH Generic Driver.....</b>	<b>232</b>
16.1	FLASH Firmware driver registers structures .....	232
16.1.1	FLASH_ProcessTypeDef .....	232
16.2	FLASH Firmware driver API description.....	232
16.2.1	FLASH peripheral features .....	232
16.2.2	How to use this driver .....	232
16.2.3	IO operation functions .....	233
16.2.4	Peripheral Control functions .....	233
16.2.5	Peripheral State functions .....	234
16.2.6	HAL_FLASH_Program .....	234
16.2.7	HAL_FLASH_Program_IT .....	234

16.2.8	HAL_FLASH_IRQHandler .....	235
16.2.9	HAL_FLASH_EndOfOperationCallback .....	235
16.2.10	HAL_FLASH_OperationErrorCallback .....	235
16.2.11	HAL_FLASH_Unlock .....	235
16.2.12	HAL_FLASH_Lock .....	235
16.2.13	HAL_FLASH_OB_Unlock .....	236
16.2.14	HAL_FLASH_OB_Lock .....	236
16.2.15	HAL_FLASH_OB_Launch .....	236
16.2.16	HAL_FLASH_GetError .....	236
16.3	FLASH Firmware driver defines .....	236
16.3.1	FLASH .....	236
<b>17</b>	<b>HAL FLASH Extension Driver .....</b>	<b>241</b>
17.1	FLASHEx Firmware driver registers structures .....	241
17.1.1	FLASH_EraseInitTypeDef .....	241
17.1.2	FLASH_OBProgramInitTypeDef .....	241
17.2	FLASHEx Firmware driver API description.....	242
17.2.1	IO operation functions .....	242
17.2.2	Peripheral Control functions .....	242
17.2.3	HAL_FLASHEx_Erase .....	242
17.2.4	HAL_FLASHEx_Erase_IT .....	243
17.2.5	HAL_FLASHEx_OBErase .....	243
17.2.6	HAL_FLASHEx_OBProgram.....	243
17.2.7	HAL_FLASHEx_OBGetConfig .....	244
17.3	FLASHEx Firmware driver defines .....	244
17.3.1	FLASHEx .....	244
<b>18</b>	<b>HAL GPIO Generic Driver.....</b>	<b>247</b>
18.1	GPIO Firmware driver registers structures .....	247
18.1.1	GPIO_InitTypeDef .....	247
18.2	GPIO Firmware driver API description .....	247
18.2.1	GPIO Peripheral features .....	247
18.2.2	How to use this driver .....	248
18.2.3	Initialization and deinitialization functions.....	248
18.2.4	IO operation functions .....	249
18.2.5	HAL_GPIO_Init.....	249
18.2.6	HAL_GPIO_DeInit .....	249
18.2.7	HAL_GPIO_ReadPin.....	249
18.2.8	HAL_GPIO_WritePin .....	250

18.2.9	HAL_GPIO_TogglePin .....	250
18.2.10	HAL_GPIO_LockPin.....	250
18.2.11	HAL_GPIO_EXTI_IRQHandler .....	251
18.2.12	HAL_GPIO_EXTI_Callback.....	251
18.3	GPIO Firmware driver defines.....	251
18.3.1	GPIO.....	251
<b>19</b>	<b>HAL GPIO Extension Driver .....</b>	<b>255</b>
19.1	GPIOEx Firmware driver API description .....	255
19.1.1	GPIO Peripheral extension features.....	255
19.1.2	How to use this driver .....	255
19.1.3	Extended features functions .....	255
19.1.4	HAL_GPIOEx_ConfigEventout.....	255
19.1.5	HAL_GPIOEx_EnableEventout.....	256
19.1.6	HAL_GPIOEx_DisableEventout .....	256
19.2	GPIOEx Firmware driver defines.....	256
19.2.1	GPIOEx .....	256
<b>20</b>	<b>HAL HCD Generic Driver .....</b>	<b>266</b>
20.1	HCD Firmware driver registers structures .....	266
20.1.1	HCD_HandleTypeDef.....	266
20.2	HCD Firmware driver API description .....	266
20.2.1	How to use this driver .....	266
20.2.2	Initialization and de-initialization functions .....	267
20.2.3	IO operation functions .....	267
20.2.4	Peripheral Control functions .....	267
20.2.5	Peripheral State functions .....	267
20.2.6	HAL_HCD_Init .....	267
20.2.7	HAL_HCD_HC_Init.....	268
20.2.8	HAL_HCD_HC_Halt .....	268
20.2.9	HAL_HCD_DeInit .....	268
20.2.10	HAL_HCD_MspInit .....	268
20.2.11	HAL_HCD_MspDeInit.....	269
20.2.12	HAL_HCD_HC_SubmitRequest.....	269
20.2.13	HAL_HCD_IRQHandler.....	269
20.2.14	HAL_HCD_SOF_Callback .....	270
20.2.15	HAL_HCD_Connect_Callback .....	270
20.2.16	HAL_HCD_Disconnect_Callback .....	270
20.2.17	HAL_HCD_HC_NotifyURBChange_Callback .....	270
20.2.18	HAL_HCD_Start .....	270

20.2.19	HAL_HCD_Stop .....	271
20.2.20	HAL_HCD_ResetPort.....	271
20.2.21	HAL_HCD_GetState.....	271
20.2.22	HAL_HCD_HC_GetURBState.....	271
20.2.23	HAL_HCD_HC_GetXferCount .....	272
20.2.24	HAL_HCD_HC_GetState .....	272
20.2.25	HAL_HCD_GetCurrentFrame .....	272
20.2.26	HAL_HCD_GetCurrentSpeed .....	272
20.3	HCD Firmware driver defines .....	272
20.3.1	HCD .....	273
<b>21</b>	<b>HAL I2C Generic Driver .....</b>	<b>274</b>
21.1	I2C Firmware driver registers structures .....	274
21.1.1	I2C_InitTypeDef.....	274
21.1.2	I2C_HandleTypeDef .....	274
21.2	I2C Firmware driver API description.....	275
21.2.1	How to use this driver .....	275
21.2.2	Initialization and de-initialization functions .....	278
21.2.3	IO operation functions .....	278
21.2.4	Peripheral State and Errors functions .....	280
21.2.5	HAL_I2C_Init .....	280
21.2.6	HAL_I2C_DeInit.....	280
21.2.7	HAL_I2C_MspInit .....	280
21.2.8	HAL_I2C_MspDeInit.....	280
21.2.9	HAL_I2C_Master_Transmit.....	281
21.2.10	HAL_I2C_Master_Receive .....	281
21.2.11	HAL_I2C_Slave_Transmit.....	281
21.2.12	HAL_I2C_Slave_Receive .....	282
21.2.13	HAL_I2C_Master_Transmit_IT.....	282
21.2.14	HAL_I2C_Master_Receive_IT.....	282
21.2.15	HAL_I2C_Slave_Transmit_IT.....	282
21.2.16	HAL_I2C_Slave_Receive_IT.....	283
21.2.17	HAL_I2C_Master_Transmit_DMA.....	283
21.2.18	HAL_I2C_Master_Receive_DMA.....	283
21.2.19	HAL_I2C_Slave_Transmit_DMA.....	284
21.2.20	HAL_I2C_Slave_Receive_DMA.....	284
21.2.21	HAL_I2C_Mem_Write.....	284
21.2.22	HAL_I2C_Mem_Read .....	285
21.2.23	HAL_I2C_Mem_Write_IT .....	285

21.2.24	HAL_I2C_Mem_Read_IT .....	285
21.2.25	HAL_I2C_Mem_Write_DMA .....	286
21.2.26	HAL_I2C_Mem_Read_DMA .....	286
21.2.27	HAL_I2C_IsDeviceReady .....	286
21.2.28	HAL_I2C_EV_IRQHandler .....	287
21.2.29	HAL_I2C_ER_IRQHandler .....	287
21.2.30	HAL_I2C_MasterTxCpltCallback .....	287
21.2.31	HAL_I2C_MasterRxCpltCallback .....	287
21.2.32	HAL_I2C_SlaveTxCpltCallback .....	288
21.2.33	HAL_I2C_SlaveRxCpltCallback .....	288
21.2.34	HAL_I2C_MemTxCpltCallback .....	288
21.2.35	HAL_I2C_MemRxCpltCallback .....	288
21.2.36	HAL_I2C_ErrorCallback .....	289
21.2.37	HAL_I2C_GetState .....	289
21.2.38	HAL_I2C_GetError .....	289
21.3	I2C Firmware driver defines .....	289
21.3.1	I2C .....	289
<b>22</b>	<b>HAL I2S Generic Driver .....</b>	<b>296</b>
22.1	I2S Firmware driver registers structures .....	296
22.1.1	I2S_InitTypeDef .....	296
22.1.2	I2S_HandleTypeDef .....	296
22.2	I2S Firmware driver API description .....	297
22.2.1	How to use this driver .....	297
22.2.2	Initialization and de-initialization functions .....	299
22.2.3	IO operation functions .....	299
22.2.4	Peripheral State and Errors functions .....	300
22.2.5	HAL_I2S_Init .....	300
22.2.6	HAL_I2S_DeInit .....	301
22.2.7	HAL_I2S_MspInit .....	301
22.2.8	HAL_I2S_MspDeInit .....	301
22.2.9	HAL_I2S_Transmit .....	301
22.2.10	HAL_I2S_Receive .....	302
22.2.11	HAL_I2S_Transmit_IT .....	302
22.2.12	HAL_I2S_Receive_IT .....	303
22.2.13	HAL_I2S_Transmit_DMA .....	303
22.2.14	HAL_I2S_Receive_DMA .....	303
22.2.15	HAL_I2S_DMAPause .....	304
22.2.16	HAL_I2S_DMAResume .....	304

22.2.17	HAL_I2S_DMAStop .....	304
22.2.18	HAL_I2S_IRQHandler .....	305
22.2.19	HAL_I2S_TxHalfCpltCallback .....	305
22.2.20	HAL_I2S_TxCpltCallback .....	305
22.2.21	HAL_I2S_RxHalfCpltCallback .....	305
22.2.22	HAL_I2S_RxCpltCallback .....	305
22.2.23	HAL_I2S_ErrorCallback .....	306
22.2.24	HAL_I2S_GetState .....	306
22.2.25	HAL_I2S_GetError .....	306
22.3	I2S Firmware driver defines .....	306
22.3.1	I2S .....	306
<b>23</b>	<b>HAL IRDA Generic Driver .....</b>	<b>311</b>
23.1	IRDA Firmware driver registers structures .....	311
23.1.1	IRDA_InitTypeDef .....	311
23.1.2	IRDA_HandleTypeDef .....	311
23.2	IRDA Firmware driver API description .....	312
23.2.1	How to use this driver .....	312
23.2.2	Initialization and Configuration functions .....	314
23.2.3	IO operation functions .....	314
23.2.4	Peripheral State and Errors functions .....	316
23.2.5	HAL_IRDA_Init .....	316
23.2.6	HAL_IRDA_DeInit .....	316
23.2.7	HAL_IRDA_MspInit .....	316
23.2.8	HAL_IRDA_MspDeInit .....	317
23.2.9	HAL_IRDA_Transmit .....	317
23.2.10	HAL_IRDA_Receive .....	317
23.2.11	HAL_IRDA_Transmit_IT .....	317
23.2.12	HAL_IRDA_Receive_IT .....	318
23.2.13	HAL_IRDA_Transmit_DMA .....	318
23.2.14	HAL_IRDA_Receive_DMA .....	318
23.2.15	HAL_IRDA_DMAPause .....	319
23.2.16	HAL_IRDA_DMAResume .....	319
23.2.17	HAL_IRDA_DMAStop .....	319
23.2.18	HAL_IRDA_IRQHandler .....	319
23.2.19	HAL_IRDA_TxCpltCallback .....	320
23.2.20	HAL_IRDA_TxHalfCpltCallback .....	320
23.2.21	HAL_IRDA_RxCpltCallback .....	320
23.2.22	HAL_IRDA_RxHalfCpltCallback .....	320

23.2.23	HAL_IRDA_ErrorCallback .....	320
23.2.24	HAL_IRDA_GetState .....	321
23.2.25	HAL_IRDA_GetError .....	321
23.3	IRDA Firmware driver defines .....	321
23.3.1	IRDA .....	321
<b>24</b>	<b>HAL IWDG Generic Driver .....</b>	<b>328</b>
24.1	IWDG Firmware driver registers structures .....	328
24.1.1	IWDG_InitTypeDef .....	328
24.1.2	IWDG_HandleTypeDef .....	328
24.2	IWDG Firmware driver API description .....	328
24.2.1	IWDG specific features .....	328
24.2.2	How to use this driver .....	329
24.2.3	Initialization and de-initialization functions .....	329
24.2.4	IO operation functions .....	330
24.2.5	Peripheral State functions .....	330
24.2.6	HAL_IWDG_Init .....	330
24.2.7	HAL_IWDG_MsplInit .....	330
24.2.8	HAL_IWDG_Start .....	330
24.2.9	HAL_IWDG_Refresh .....	331
24.2.10	HAL_IWDG_GetState .....	331
24.3	IWDG Firmware driver defines .....	331
24.3.1	IWDG .....	331
<b>25</b>	<b>HAL NAND Generic Driver .....</b>	<b>334</b>
25.1	NAND Firmware driver registers structures .....	334
25.1.1	NAND_IDTypeDef .....	334
25.1.2	NAND_AddressTypeDef .....	334
25.1.3	NAND_InfoTypeDef .....	334
25.1.4	NAND_HandleTypeDef .....	335
25.2	NAND Firmware driver API description .....	335
25.2.1	How to use this driver .....	335
25.2.2	NAND Initialization and de-initialization functions .....	336
25.2.3	NAND Input and Output functions .....	336
25.2.4	NAND Control functions .....	336
25.2.5	NAND State functions .....	337
25.2.6	HAL_NAND_Init .....	337
25.2.7	HAL_NAND_DeInit .....	337
25.2.8	HAL_NAND_MsplInit .....	337



25.2.9	HAL_NAND_MspDeInit .....	337
25.2.10	HAL_NAND_IRQHandler .....	338
25.2.11	HAL_NAND_ITCallback .....	338
25.2.12	HAL_NAND_Read_ID .....	338
25.2.13	HAL_NAND_Reset .....	338
25.2.14	HAL_NAND_Read_Page .....	339
25.2.15	HAL_NAND_Write_Page.....	339
25.2.16	HAL_NAND_Read_SpareArea .....	339
25.2.17	HAL_NAND_Write_SpareArea.....	339
25.2.18	HAL_NAND_Erase_Block .....	340
25.2.19	HAL_NAND_Read_Status .....	340
25.2.20	HAL_NAND_Address_Inc .....	340
25.2.21	HAL_NAND_ECC_Enable .....	341
25.2.22	HAL_NAND_ECC_Disable.....	341
25.2.23	HAL_NAND_GetECC .....	341
25.2.24	HAL_NAND_GetState .....	341
25.2.25	HAL_NAND_Read_Status .....	342
25.3	NAND Firmware driver defines.....	342
25.3.1	NAND.....	342
<b>26</b>	<b>HAL NOR Generic Driver.....</b>	<b>344</b>
26.1	NOR Firmware driver registers structures .....	344
26.1.1	NOR_IDTypeDef .....	344
26.1.2	NOR_CFITypeDef .....	344
26.1.3	NOR_HandleTypeDef.....	344
26.2	NOR Firmware driver API description .....	345
26.2.1	How to use this driver .....	345
26.2.2	NOR Initialization and de_initialization functions .....	346
26.2.3	NOR Input and Output functions .....	346
26.2.4	NOR Control functions.....	346
26.2.5	NOR State functions.....	346
26.2.6	HAL_NOR_Init.....	346
26.2.7	HAL_NOR_DeInit .....	347
26.2.8	HAL_NOR_MspInit .....	347
26.2.9	HAL_NOR_MspDeInit .....	347
26.2.10	HAL_NOR_MspWait.....	347
26.2.11	HAL_NOR_Read_ID .....	348
26.2.12	HAL_NOR_ReturnToReadMode.....	348
26.2.13	HAL_NOR_Read .....	348

26.2.14	HAL_NOR_Program .....	348
26.2.15	HAL_NOR_ReadBuffer .....	349
26.2.16	HAL_NOR_ProgramBuffer .....	349
26.2.17	HAL_NOR_Erase_Block .....	349
26.2.18	HAL_NOR_Erase_Chip .....	350
26.2.19	HAL_NOR_Read_CFI .....	350
26.2.20	HAL_NOR_WriteOperation_Enable .....	350
26.2.21	HAL_NOR_WriteOperation_Disable .....	350
26.2.22	HAL_NOR_GetState .....	351
26.2.23	HAL_NOR_GetStatus .....	351
26.3	NOR Firmware driver defines .....	351
26.3.1	NOR .....	351
<b>27</b>	<b>HAL PCCARD Generic Driver .....</b>	<b>354</b>
27.1	PCCARD Firmware driver registers structures .....	354
27.1.1	PCCARD_HandleTypeDef .....	354
27.2	PCCARD Firmware driver API description .....	354
27.2.1	How to use this driver .....	354
27.2.2	PCCARD Initialization and de-initialization functions .....	355
27.2.3	PCCARD Input Output and memory functions .....	355
27.2.4	PCCARD Peripheral State functions .....	355
27.2.5	HAL_PCCARD_Init .....	355
27.2.6	HAL_PCCARD_DeInit .....	356
27.2.7	HAL_PCCARD_MspInit .....	356
27.2.8	HAL_PCCARD_MspDeInit .....	356
27.2.9	HAL_CF_Read_ID .....	356
27.2.10	HAL_CF_Read_Sector .....	357
27.2.11	HAL_CF_Write_Sector .....	357
27.2.12	HAL_CF_Erase_Sector .....	357
27.2.13	HAL_CF_Reset .....	358
27.2.14	HAL_PCCARD_IRQHandler .....	358
27.2.15	HAL_PCCARD_ITCallback .....	358
27.2.16	HAL_PCCARD_GetState .....	358
27.2.17	HAL_CF_GetStatus .....	359
27.2.18	HAL_CF_ReadStatus .....	359
27.3	PCCARD Firmware driver defines .....	359
27.3.1	PCCARD .....	359
<b>28</b>	<b>HAL PCD Generic Driver .....</b>	<b>361</b>
28.1	PCD Firmware driver registers structures .....	361

28.1.1	PCD_HandleTypeDef .....	361
28.2	PCD Firmware driver API description .....	361
28.2.1	How to use this driver .....	361
28.2.2	Initialization and de-initialization functions .....	362
28.2.3	IO operation functions .....	362
28.2.4	Peripheral Control functions .....	362
28.2.5	Peripheral State functions .....	363
28.2.6	HAL_PCD_Init .....	363
28.2.7	HAL_PCD_DeInit .....	363
28.2.8	HAL_PCD_MspInit .....	363
28.2.9	HAL_PCD_MspDeInit .....	363
28.2.10	HAL_PCD_Start .....	364
28.2.11	HAL_PCD_Stop .....	364
28.2.12	HAL_PCD_IRQHandler .....	364
28.2.13	HAL_PCD_DataOutStageCallback .....	364
28.2.14	HAL_PCD_DataInStageCallback .....	364
28.2.15	HAL_PCD_SetupStageCallback .....	365
28.2.16	HAL_PCD_SOFCallback .....	365
28.2.17	HAL_PCD_ResetCallback .....	365
28.2.18	HAL_PCD_SuspendCallback .....	365
28.2.19	HAL_PCD_ResumeCallback .....	365
28.2.20	HAL_PCD_ISOOUTIncompleteCallback .....	366
28.2.21	HAL_PCD_ISOINIncompleteCallback .....	366
28.2.22	HAL_PCD_ConnectCallback .....	366
28.2.23	HAL_PCD_DisconnectCallback .....	366
28.2.24	HAL_PCD_DevConnect .....	366
28.2.25	HAL_PCD_DevDisconnect .....	367
28.2.26	HAL_PCD_SetAddress .....	367
28.2.27	HAL_PCD_EP_Open .....	367
28.2.28	HAL_PCD_EP_Close .....	367
28.2.29	HAL_PCD_EP_Receive .....	368
28.2.30	HAL_PCD_EP_GetRxCount .....	368
28.2.31	HAL_PCD_EP_Transmit .....	368
28.2.32	HAL_PCD_EP_SetStall .....	368
28.2.33	HAL_PCD_EP_ClrStall .....	369
28.2.34	HAL_PCD_EP_Flush .....	369
28.2.35	HAL_PCD_ActiveRemoteWakeup .....	369
28.2.36	HAL_PCD_DeActiveRemoteWakeup .....	369

28.2.37	HAL_PCD_GetState .....	369
28.3	PCD Firmware driver defines .....	370
28.3.1	PCD .....	370
<b>29</b>	<b>HAL PCD Extension Driver .....</b>	<b>379</b>
29.1	PCDEx Firmware driver API description .....	379
29.1.1	Extended Peripheral Control functions .....	379
29.1.2	HAL_PCDEx_PMAConfig .....	379
29.1.3	HAL_PCDEx_SetConnectionState .....	379
29.2	PCDEx Firmware driver defines .....	379
29.2.1	PCDEx .....	380
<b>30</b>	<b>HAL PWR Generic Driver .....</b>	<b>381</b>
30.1	PWR Firmware driver registers structures .....	381
30.1.1	PWR_PVDTypeDef .....	381
30.2	PWR Firmware driver API description .....	381
30.2.1	Initialization and de-initialization functions .....	381
30.2.2	Peripheral Control functions .....	381
30.2.3	HAL_PWR_DeInit .....	383
30.2.4	HAL_PWR_EnableBkUpAccess .....	384
30.2.5	HAL_PWR_DisableBkUpAccess .....	384
30.2.6	HAL_PWR_ConfigPVD .....	384
30.2.7	HAL_PWR_EnablePVD .....	384
30.2.8	HAL_PWR_DisablePVD .....	384
30.2.9	HAL_PWR_EnableWakeUpPin .....	385
30.2.10	HAL_PWR_DisableWakeUpPin .....	385
30.2.11	HAL_PWR_EnterSLEEPMode .....	385
30.2.12	HAL_PWR_EnterSTOPMode .....	385
30.2.13	HAL_PWR_EnterSTANDBYMode .....	386
30.2.14	HAL_PWR_EnableSleepOnExit .....	386
30.2.15	HAL_PWR_DisableSleepOnExit .....	387
30.2.16	HAL_PWR_EnableSEVOnPend .....	387
30.2.17	HAL_PWR_DisableSEVOnPend .....	387
30.2.18	HAL_PWR_PVD_IRQHandler .....	387
30.2.19	HAL_PWR_PVDCallback .....	387
30.3	PWR Firmware driver defines .....	388
30.3.1	PWR .....	388
<b>31</b>	<b>HAL RCC Generic Driver .....</b>	<b>393</b>
31.1	RCC Firmware driver registers structures .....	393

31.1.1	RCC_PLLInitTypeDef .....	393
31.1.2	RCC_ClkInitTypeDef .....	393
<b>31.2</b>	<b>RCC Firmware driver API description .....</b>	<b>394</b>
31.2.1	RCC specific features .....	394
31.2.2	RCC Limitations.....	394
31.2.3	Initialization and de-initialization functions .....	394
31.2.4	Peripheral Control functions .....	395
31.2.5	HAL_RCC_DeInit .....	396
31.2.6	HAL_RCC_OscConfig .....	396
31.2.7	HAL_RCC_ClockConfig .....	396
31.2.8	HAL_RCC_MCOConfig.....	397
31.2.9	HAL_RCC_EnableCSS .....	398
31.2.10	HAL_RCC_DisableCSS .....	398
31.2.11	HAL_RCC_GetSysClockFreq .....	398
31.2.12	HAL_RCC_GetHCLKFreq .....	399
31.2.13	HAL_RCC_GetPCLK1Freq .....	399
31.2.14	HAL_RCC_GetPCLK2Freq .....	399
31.2.15	HAL_RCC_GetOscConfig .....	400
31.2.16	HAL_RCC_GetClockConfig .....	400
31.2.17	HAL_RCC_NMI_IRQHandler .....	400
31.2.18	HAL_RCC_CSSCallback.....	400
<b>31.3</b>	<b>RCC Firmware driver defines .....</b>	<b>400</b>
31.3.1	RCC .....	401
<b>32</b>	<b>HAL RCC Extension Driver .....</b>	<b>414</b>
<b>32.1</b>	<b>RCCEX Firmware driver registers structures .....</b>	<b>414</b>
32.1.1	RCC_OscInitTypeDef .....	414
32.1.2	RCC_PeriphCLKInitTypeDef .....	414
<b>32.2</b>	<b>RCCEX Firmware driver API description .....</b>	<b>415</b>
32.2.1	Extended Peripheral Control functions.....	415
32.2.2	HAL_RCCEX_PeriphCLKConfig.....	415
32.2.3	HAL_RCCEX_GetPeriphCLKConfig.....	416
32.2.4	HAL_RCCEX_GetPeriphCLKFreq.....	416
<b>32.3</b>	<b>RCCEX Firmware driver defines.....</b>	<b>416</b>
32.3.1	RCCEX.....	416
<b>33</b>	<b>HAL RTC Generic Driver .....</b>	<b>425</b>
<b>33.1</b>	<b>RTC Firmware driver registers structures .....</b>	<b>425</b>
33.1.1	RTC_TimeTypeDef.....	425

33.1.2	RTC_AlarmTypeDef .....	425
33.1.3	RTC_InitTypeDef .....	425
33.1.4	RTC_DateTypeDef .....	426
33.1.5	RTC_HandleTypeDef .....	426
33.2	RTC Firmware driver API description .....	427
33.2.1	How to use this driver .....	427
33.2.2	WARNING: Drivers Restrictions .....	427
33.2.3	Backup Domain Operating Condition .....	428
33.2.4	Backup Domain Reset .....	428
33.2.5	Backup Domain Access .....	428
33.2.6	RTC and low power modes .....	429
33.2.7	Initialization and de-initialization functions .....	429
33.2.8	RTC Time and Date functions .....	429
33.2.9	RTC Alarm functions .....	429
33.2.10	Peripheral State functions .....	430
33.2.11	Peripheral Control functions .....	430
33.2.12	HAL_RTC_Init .....	430
33.2.13	HAL_RTC_DeInit .....	430
33.2.14	HAL_RTC_MspInit .....	430
33.2.15	HAL_RTC_MspDeInit .....	430
33.2.16	HAL_RTC_SetTime .....	431
33.2.17	HAL_RTC_GetTime .....	431
33.2.18	HAL_RTC_SetDate .....	431
33.2.19	HAL_RTC_GetDate .....	432
33.2.20	HAL_RTC_SetAlarm .....	432
33.2.21	HAL_RTC_SetAlarm_IT .....	432
33.2.22	HAL_RTC_GetAlarm .....	433
33.2.23	HAL_RTC_DeactivateAlarm .....	433
33.2.24	HAL_RTC_AlarmIRQHandler .....	433
33.2.25	HAL_RTC_AlarmAEventCallback .....	433
33.2.26	HAL_RTC_PollForAlarmAEvent .....	434
33.2.27	HAL_RTC_GetState .....	434
33.2.28	HAL_RTC_WaitForSynchro .....	434
33.3	RTC Firmware driver defines .....	434
33.3.1	RTC .....	435
<b>34</b>	<b>HAL RTC Extension Driver .....</b>	<b>442</b>
34.1	RTCEX Firmware driver registers structures .....	442
34.1.1	RTC_TamperTypeDef .....	442

34.2	RTCEx Firmware driver API description.....	442
34.2.1	RTC Tamper functions .....	442
34.2.2	RTC Second functions.....	442
34.2.3	Extension Peripheral Control functions .....	442
34.2.4	HAL_RTCEx_SetTamper .....	443
34.2.5	HAL_RTCEx_SetTamper_IT .....	443
34.2.6	HAL_RTCEx_DeactivateTamper .....	443
34.2.7	HAL_RTCEx_TamperIRQHandler .....	444
34.2.8	HAL_RTCEx_Tamper1EventCallback .....	444
34.2.9	HAL_RTCEx_PollForTamper1Event.....	444
34.2.10	HAL_RTCEx_SetSecond_IT .....	444
34.2.11	HAL_RTCEx_DeactivateSecond.....	444
34.2.12	HAL_RTCEx_RTCIRQHandler .....	445
34.2.13	HAL_RTCEx_RTCEventCallback .....	445
34.2.14	HAL_RTCEx_RTCEventErrorCallback .....	445
34.2.15	HAL_RTCEx_BKUPWrite.....	445
34.2.16	HAL_RTCEx_BKUPRead .....	446
34.2.17	HAL_RTCEx_SetSmoothCalib.....	446
34.3	RTCEx Firmware driver defines .....	446
34.3.1	RTCEx .....	446
<b>35</b>	<b>HAL SD Generic Driver .....</b>	<b>454</b>
35.1	SD Firmware driver registers structures.....	454
35.1.1	SD_HandleTypeDef.....	454
35.1.2	HAL_SD_CSDTypeDef .....	454
35.1.3	HAL_SD_CIDTypeDef .....	456
35.1.4	HAL_SD_CardStatusTypeDef .....	457
35.1.5	HAL_SD_CardInfoTypeDef.....	458
35.2	SD Firmware driver API description .....	458
35.2.1	How to use this driver .....	458
35.2.2	Initialization and de-initialization functions .....	460
35.2.3	IO operation functions .....	460
35.2.4	Peripheral Control functions .....	461
35.2.5	Peripheral State functions .....	461
35.2.6	HAL_SD_Init.....	461
35.2.7	HAL_SD_DeInit .....	461
35.2.8	HAL_SD_MspInit .....	461
35.2.9	HAL_SD_MspDeInit .....	462
35.2.10	HAL_SD_ReadBlocks .....	462

35.2.11	HAL_SD_WriteBlocks.....	462
35.2.12	HAL_SD_ReadBlocks_DMA .....	463
35.2.13	HAL_SD_WriteBlocks_DMA .....	463
35.2.14	HAL_SD_CheckReadOperation.....	463
35.2.15	HAL_SD_CheckWriteOperation .....	464
35.2.16	HAL_SD_Erase .....	464
35.2.17	HAL_SD_IRQHandler.....	464
35.2.18	HAL_SD_XferCpltCallback.....	464
35.2.19	HAL_SD_XferErrorCallback .....	464
35.2.20	HAL_SD_DMA_RxCpltCallback.....	465
35.2.21	HAL_SD_DMA_RxErrorCallback .....	465
35.2.22	HAL_SD_DMA_TxCpltCallback .....	465
35.2.23	HAL_SD_DMA_TxErrorCallback.....	465
35.2.24	HAL_SD_Get_CardInfo .....	465
35.2.25	HAL_SD_WideBusOperation_Config.....	466
35.2.26	HAL_SD_StopTransfer.....	466
35.2.27	HAL_SD_HighSpeed.....	466
35.2.28	HAL_SD_SendSDStatus .....	467
35.2.29	HAL_SD_GetStatus.....	467
35.2.30	HAL_SD_GetCardStatus.....	467
35.3	SD Firmware driver defines.....	467
35.3.1	SD .....	467
<b>36</b>	<b>HAL SMARTCARD Generic Driver.....</b>	<b>481</b>
36.1	SMARTCARD Firmware driver registers structures .....	481
36.1.1	SMARTCARD_InitTypeDef .....	481
36.1.2	SMARTCARD_HandleTypeDef.....	482
36.2	SMARTCARD Firmware driver API description.....	483
36.2.1	How to use this driver .....	483
36.2.2	Initialization and Configuration functions.....	485
36.2.3	IO operation functions .....	485
36.2.4	Peripheral State and Errors functions .....	487
36.2.5	HAL_SMARTCARD_Init.....	487
36.2.6	HAL_SMARTCARD_DeInit .....	487
36.2.7	HAL_SMARTCARD_MspInit .....	488
36.2.8	HAL_SMARTCARD_MspDeInit .....	488
36.2.9	HAL_SMARTCARD_Transmit.....	488
36.2.10	HAL_SMARTCARD_Receive.....	488
36.2.11	HAL_SMARTCARD_Transmit_IT .....	489



36.2.12	HAL_SMARTCARD_Receive_IT .....	489
36.2.13	HAL_SMARTCARD_Transmit_DMA.....	489
36.2.14	HAL_SMARTCARD_Receive_DMA.....	490
36.2.15	HAL_SMARTCARD_IRQHandler.....	490
36.2.16	HAL_SMARTCARD_TxCpltCallback .....	490
36.2.17	HAL_SMARTCARD_RxCpltCallback .....	491
36.2.18	HAL_SMARTCARD_ErrorCallback.....	491
36.2.19	HAL_SMARTCARD_GetState .....	491
36.2.20	HAL_SMARTCARD_GetError .....	491
36.3	SMARTCARD Firmware driver defines .....	492
36.3.1	SMARTCARD.....	492
<b>37</b>	<b>HAL SPI Generic Driver.....</b>	<b>502</b>
37.1	SPI Firmware driver registers structures .....	502
37.1.1	SPI_InitTypeDef .....	502
37.1.2	__SPI_HandleTypeDef.....	503
37.2	SPI Firmware driver API description .....	503
37.2.1	How to use this driver .....	503
37.2.2	Initialization and de-initialization functions .....	505
37.2.3	IO operation functions .....	506
37.2.4	Peripheral State and Errors functions .....	506
37.2.5	HAL_SPI_Init .....	507
37.2.6	HAL_SPI_DeInit .....	507
37.2.7	HAL_SPI_MspInit .....	507
37.2.8	HAL_SPI_MspDeInit.....	507
37.2.9	HAL_SPI_Transmit.....	508
37.2.10	HAL_SPI_Receive.....	508
37.2.11	HAL_SPI_TransmitReceive.....	508
37.2.12	HAL_SPI_Transmit_IT.....	508
37.2.13	HAL_SPI_Receive_IT.....	509
37.2.14	HAL_SPI_TransmitReceive_IT .....	509
37.2.15	HAL_SPI_Transmit_DMA.....	509
37.2.16	HAL_SPI_Receive_DMA.....	510
37.2.17	HAL_SPI_TransmitReceive_DMA.....	510
37.2.18	HAL_SPI_DMAPause.....	510
37.2.19	HAL_SPI_DMAResume .....	510
37.2.20	HAL_SPI_DMAStop .....	511
37.2.21	HAL_SPI_IRQHandler.....	511
37.2.22	HAL_SPI_TxCpltCallback .....	511

37.2.23	HAL_SPI_RxCpltCallback .....	511
37.2.24	HAL_SPI_TxRxCpltCallback .....	511
37.2.25	HAL_SPI_TxHalfCpltCallback .....	512
37.2.26	HAL_SPI_RxHalfCpltCallback .....	512
37.2.27	HAL_SPI_TxRxHalfCpltCallback .....	512
37.2.28	HAL_SPI_ErrorCallback .....	512
37.2.29	HAL_SPI_GetState .....	513
37.2.30	HAL_SPI_GetError .....	513
37.3	SPI Firmware driver defines .....	513
37.3.1	SPI .....	513
<b>38</b>	<b>HAL SRAM Generic Driver .....</b>	<b>522</b>
38.1	SRAM Firmware driver registers structures .....	522
38.1.1	SRAM_HandleTypeDef .....	522
38.2	SRAM Firmware driver API description .....	522
38.2.1	How to use this driver .....	522
38.2.2	SRAM Initialization and de_initialization functions .....	523
38.2.3	SRAM Input and Output functions .....	523
38.2.4	SRAM Control functions .....	523
38.2.5	SRAM State functions .....	524
38.2.6	HAL_SRAM_Init .....	524
38.2.7	HAL_SRAM_DeInit .....	524
38.2.8	HAL_SRAM_MspInit .....	524
38.2.9	HAL_SRAM_MspDeInit .....	524
38.2.10	HAL_SRAM_DMA_XferCpltCallback .....	525
38.2.11	HAL_SRAM_DMA_XferErrorCallback .....	525
38.2.12	HAL_SRAM_Read_8b .....	525
38.2.13	HAL_SRAM_Write_8b .....	525
38.2.14	HAL_SRAM_Read_16b .....	526
38.2.15	HAL_SRAM_Write_16b .....	526
38.2.16	HAL_SRAM_Read_32b .....	526
38.2.17	HAL_SRAM_Write_32b .....	527
38.2.18	HAL_SRAM_Read_DMA .....	527
38.2.19	HAL_SRAM_Write_DMA .....	527
38.2.20	HAL_SRAM_WriteOperation_Enable .....	528
38.2.21	HAL_SRAM_WriteOperation_Disable .....	528
38.2.22	HAL_SRAM_GetState .....	528
38.3	SRAM Firmware driver defines .....	528
38.3.1	SRAM .....	528

<b>39</b>	<b>HAL TIM Generic Driver .....</b>	<b>529</b>
39.1	TIM Firmware driver registers structures.....	529
39.1.1	TIM_Base_InitTypeDef.....	529
39.1.2	TIM_OC_InitTypeDef.....	529
39.1.3	TIM_OnePulse_InitTypeDef .....	530
39.1.4	TIM_IC_InitTypeDef .....	531
39.1.5	TIM_Encoder_InitTypeDef .....	531
39.1.6	TIM_ClockConfigTypeDef .....	532
39.1.7	TIM_ClearInputConfigTypeDef.....	532
39.1.8	TIM_SlaveConfigTypeDef .....	533
39.1.9	TIM_HandleTypeDef .....	533
39.2	TIM Firmware driver API description .....	534
39.2.1	TIMER Generic features.....	534
39.2.2	How to use this driver .....	534
39.2.3	Time Base functions .....	535
39.2.4	Time Output Compare functions .....	535
39.2.5	Time PWM functions .....	536
39.2.6	Time Input Capture functions .....	536
39.2.7	Time One Pulse functions .....	537
39.2.8	Time Encoder functions.....	537
39.2.9	IRQ handler management .....	538
39.2.10	Peripheral Control functions .....	538
39.2.11	TIM Callbacks functions .....	538
39.2.12	Peripheral State functions .....	539
39.2.13	HAL_TIM_Base_Init .....	539
39.2.14	HAL_TIM_Base_DeInit.....	539
39.2.15	HAL_TIM_Base_MspInit.....	539
39.2.16	HAL_TIM_Base_MspDeInit.....	539
39.2.17	HAL_TIM_Base_Start.....	540
39.2.18	HAL_TIM_Base_Stop.....	540
39.2.19	HAL_TIM_Base_Start_IT .....	540
39.2.20	HAL_TIM_Base_Stop_IT.....	540
39.2.21	HAL_TIM_Base_Start_DMA .....	540
39.2.22	HAL_TIM_Base_Stop_DMA.....	541
39.2.23	HAL_TIM_OC_Init .....	541
39.2.24	HAL_TIM_OC_DeInit.....	541
39.2.25	HAL_TIM_OC_MspInit .....	541
39.2.26	HAL_TIM_OC_MspDeInit.....	542

39.2.27	HAL_TIM_OC_Start .....	542
39.2.28	HAL_TIM_OC_Stop.....	542
39.2.29	HAL_TIM_OC_Start_IT .....	542
39.2.30	HAL_TIM_OC_Stop_IT .....	543
39.2.31	HAL_TIM_OC_Start_DMA .....	543
39.2.32	HAL_TIM_OC_Stop_DMA .....	543
39.2.33	HAL_TIM_PWM_Init.....	544
39.2.34	HAL_TIM_PWM_DeInit .....	544
39.2.35	HAL_TIM_PWM_MspInit.....	544
39.2.36	HAL_TIM_PWM_MspDeInit .....	544
39.2.37	HAL_TIM_PWM_Start.....	544
39.2.38	HAL_TIM_PWM_Stop .....	545
39.2.39	HAL_TIM_PWM_Start_IT .....	545
39.2.40	HAL_TIM_PWM_Stop_IT .....	545
39.2.41	HAL_TIM_PWM_Start_DMA .....	546
39.2.42	HAL_TIM_PWM_Stop_DMA .....	546
39.2.43	HAL_TIM_IC_Init .....	546
39.2.44	HAL_TIM_IC_DeInit .....	547
39.2.45	HAL_TIM_IC_MspInit .....	547
39.2.46	HAL_TIM_IC_MspDeInit.....	547
39.2.47	HAL_TIM_IC_Start .....	547
39.2.48	HAL_TIM_IC_Stop .....	547
39.2.49	HAL_TIM_IC_Start_IT .....	548
39.2.50	HAL_TIM_IC_Stop_IT .....	548
39.2.51	HAL_TIM_IC_Start_DMA .....	548
39.2.52	HAL_TIM_IC_Stop_DMA .....	549
39.2.53	HAL_TIM_OnePulse_Init.....	549
39.2.54	HAL_TIM_OnePulse_DeInit .....	549
39.2.55	HAL_TIM_OnePulse_MspInit.....	549
39.2.56	HAL_TIM_OnePulse_MspDeInit .....	550
39.2.57	HAL_TIM_OnePulse_Start .....	550
39.2.58	HAL_TIM_OnePulse_Stop .....	550
39.2.59	HAL_TIM_OnePulse_Start_IT .....	550
39.2.60	HAL_TIM_OnePulse_Stop_IT .....	551
39.2.61	HAL_TIM_Encoder_Init .....	551
39.2.62	HAL_TIM_Encoder_DeInit .....	551
39.2.63	HAL_TIM_Encoder_MspInit .....	551
39.2.64	HAL_TIM_Encoder_MspDeInit.....	552
39.2.65	HAL_TIM_Encoder_Start .....	552

39.2.66	HAL_TIM_Encoder_Stop .....	552
39.2.67	HAL_TIM_Encoder_Start_IT .....	552
39.2.68	HAL_TIM_Encoder_Stop_IT .....	553
39.2.69	HAL_TIM_Encoder_Start_DMA .....	553
39.2.70	HAL_TIM_Encoder_Stop_DMA .....	553
39.2.71	HAL_TIM_IRQHandler .....	554
39.2.72	HAL_TIM_OC_ConfigChannel .....	554
39.2.73	HAL_TIM_IC_ConfigChannel .....	554
39.2.74	HAL_TIM_PWM_ConfigChannel .....	555
39.2.75	HAL_TIM_OnePulse_ConfigChannel .....	555
39.2.76	HAL_TIM_DMABurst_WriteStart .....	555
39.2.77	HAL_TIM_DMABurst_WriteStop .....	556
39.2.78	HAL_TIM_DMABurst_ReadStart .....	556
39.2.79	HAL_TIM_DMABurst_ReadStop .....	557
39.2.80	HAL_TIM_GenerateEvent .....	557
39.2.81	HAL_TIM_ConfigOCrefClear .....	558
39.2.82	HAL_TIM_ConfigClockSource .....	558
39.2.83	HAL_TIM_ConfigTI1Input .....	558
39.2.84	HAL_TIM_SlaveConfigSynchronization .....	559
39.2.85	HAL_TIM_SlaveConfigSynchronization_IT .....	559
39.2.86	HAL_TIM_ReadCapturedValue .....	559
39.2.87	HAL_TIM_PeriodElapsedCallback .....	560
39.2.88	HAL_TIM_OC_DelayElapsedCallback .....	560
39.2.89	HAL_TIM_IC_CaptureCallback .....	560
39.2.90	HAL_TIM_PWM_PulseFinishedCallback .....	560
39.2.91	HAL_TIM_TriggerCallback .....	560
39.2.92	HAL_TIM_ErrorCallback .....	561
39.2.93	HAL_TIM_Base_GetState .....	561
39.2.94	HAL_TIM_OC_GetState .....	561
39.2.95	HAL_TIM_PWM_GetState .....	561
39.2.96	HAL_TIM_IC_GetState .....	561
39.2.97	HAL_TIM_OnePulse_GetState .....	562
39.2.98	HAL_TIM_Encoder_GetState .....	562
39.3	TIM Firmware driver defines .....	562
39.3.1	TIM .....	562
<b>40</b>	<b>HAL TIM Extension Driver .....</b>	<b>582</b>
40.1	TIMEx Firmware driver registers structures .....	582
40.1.1	TIM_HallSensor_InitTypeDef .....	582

40.1.2	TIM_BreakDeadTimeConfigTypeDef .....	582
40.1.3	TIM_MasterConfigTypeDef .....	583
40.2	<b>TIMEx Firmware driver API description .....</b>	<b>583</b>
40.2.1	TIMER Extended features .....	583
40.2.2	How to use this driver .....	583
40.2.3	Timer Hall Sensor functions .....	584
40.2.4	Timer Complementary Output Compare functions.....	585
40.2.5	Timer Complementary PWM functions.....	585
40.2.6	Timer Complementary One Pulse functions.....	585
40.2.7	Peripheral Control functions .....	586
40.2.8	Extension Callbacks functions.....	586
40.2.9	Extension Peripheral State functions .....	586
40.2.10	HAL_TIMEx_HallSensor_Init.....	586
40.2.11	HAL_TIMEx_HallSensor_DeInit .....	586
40.2.12	HAL_TIMEx_HallSensor_MspInit.....	587
40.2.13	HAL_TIMEx_HallSensor_MspDeInit .....	587
40.2.14	HAL_TIMEx_HallSensor_Start.....	587
40.2.15	HAL_TIMEx_HallSensor_Stop .....	587
40.2.16	HAL_TIMEx_HallSensor_Start_IT .....	588
40.2.17	HAL_TIMEx_HallSensor_Stop_IT .....	588
40.2.18	HAL_TIMEx_HallSensor_Start_DMA.....	588
40.2.19	HAL_TIMEx_HallSensor_Stop_DMA .....	588
40.2.20	HAL_TIMEx_OCN_Start.....	588
40.2.21	HAL_TIMEx_OCN_Stop.....	589
40.2.22	HAL_TIMEx_OCN_Start_IT .....	589
40.2.23	HAL_TIMEx_OCN_Stop_IT .....	589
40.2.24	HAL_TIMEx_OCN_Start_DMA .....	590
40.2.25	HAL_TIMEx_OCN_Stop_DMA.....	590
40.2.26	HAL_TIMEx_PWMN_Start .....	590
40.2.27	HAL_TIMEx_PWMN_Stop .....	591
40.2.28	HAL_TIMEx_PWMN_Start_IT .....	591
40.2.29	HAL_TIMEx_PWMN_Stop_IT .....	591
40.2.30	HAL_TIMEx_PWMN_Start_DMA .....	592
40.2.31	HAL_TIMEx_PWMN_Stop_DMA .....	592
40.2.32	HAL_TIMEx_OnePulseN_Start .....	592
40.2.33	HAL_TIMEx_OnePulseN_Stop .....	593
40.2.34	HAL_TIMEx_OnePulseN_Start_IT .....	593
40.2.35	HAL_TIMEx_OnePulseN_Stop_IT .....	593
40.2.36	HAL_TIMEx_ConfigCommutationEvent .....	593

40.2.37	HAL_TIMEx_ConfigCommutationEvent_IT .....	594
40.2.38	HAL_TIMEx_ConfigCommutationEvent_DMA .....	595
40.2.39	HAL_TIMEx_ConfigBreakDeadTime .....	595
40.2.40	HAL_TIMEx_MasterConfigSynchronization .....	596
40.2.41	HAL_TIMEx_CommutationCallback .....	596
40.2.42	HAL_TIMEx_BreakCallback .....	596
40.2.43	TIMEx_DMABreakCommutationCplt .....	596
40.2.44	HAL_TIMEx_HallSensor_GetState .....	597
40.3	TIMEx Firmware driver defines .....	597
40.3.1	TIMEx .....	597
<b>41</b>	<b>HAL UART Generic Driver.....</b>	<b>598</b>
41.1	UART Firmware driver registers structures .....	598
41.1.1	UART_InitTypeDef .....	598
41.1.2	UART_HandleTypeDef .....	598
41.2	UART Firmware driver API description .....	599
41.2.1	How to use this driver .....	599
41.2.2	Initialization and Configuration functions .....	601
41.2.3	IO operation functions .....	602
41.2.4	Peripheral Control functions .....	603
41.2.5	Peripheral State and Errors functions .....	603
41.2.6	HAL_UART_Init .....	604
41.2.7	HAL_HalfDuplex_Init .....	604
41.2.8	HAL_LIN_Init .....	604
41.2.9	HAL_MultiProcessor_Init .....	605
41.2.10	HAL_UART_DeInit .....	605
41.2.11	HAL_UART_MspInit .....	605
41.2.12	HAL_UART_MspDeInit .....	605
41.2.13	HAL_UART_Transmit .....	606
41.2.14	HAL_UART_Receive .....	606
41.2.15	HAL_UART_Transmit_IT .....	606
41.2.16	HAL_UART_Receive_IT .....	607
41.2.17	HAL_UART_Transmit_DMA .....	607
41.2.18	HAL_UART_Receive_DMA .....	607
41.2.19	HAL_UART_DMAPause .....	607
41.2.20	HAL_UART_DMAResume .....	608
41.2.21	HAL_UART_DMAStop .....	608
41.2.22	HAL_UART_IRQHandler .....	608
41.2.23	HAL_UART_TxCpltCallback .....	608

41.2.24	HAL_UART_TxHalfCpltCallback .....	609
41.2.25	HAL_UART_RxCpltCallback .....	609
41.2.26	HAL_UART_RxHalfCpltCallback.....	609
41.2.27	HAL_UART_ErrorCallback.....	609
41.2.28	HAL_LIN_SendBreak .....	610
41.2.29	HAL_MultiProcessor_EnterMuteMode .....	610
41.2.30	HAL_MultiProcessor_ExitMuteMode.....	610
41.2.31	HAL_HalfDuplex_EnableTransmitter .....	610
41.2.32	HAL_HalfDuplex_EnableReceiver .....	611
41.2.33	HAL_UART_GetState.....	611
41.2.34	HAL_UART_GetError .....	611
41.3	UART Firmware driver defines .....	611
41.3.1	UART .....	611
<b>42</b>	<b>HAL USART Generic Driver .....</b>	<b>621</b>
42.1	USART Firmware driver registers structures.....	621
42.1.1	USART_InitTypeDef .....	621
42.1.2	USART_HandleTypeDef .....	621
42.2	USART Firmware driver API description .....	622
42.2.1	How to use this driver .....	622
42.2.2	Initialization and Configuration functions.....	624
42.2.3	IO operation functions .....	625
42.2.4	Peripheral State and Errors functions .....	626
42.2.5	HAL_USART_Init.....	626
42.2.6	HAL_USART_DeInit .....	627
42.2.7	HAL_USART_MspInit.....	627
42.2.8	HAL_USART_MspDeInit .....	627
42.2.9	HAL_USART_Transmit .....	627
42.2.10	HAL_USART_Receive .....	628
42.2.11	HAL_USART_TransmitReceive .....	628
42.2.12	HAL_USART_Transmit_IT .....	628
42.2.13	HAL_USART_Receive_IT .....	629
42.2.14	HAL_USART_TransmitReceive_IT .....	629
42.2.15	HAL_USART_Transmit_DMA .....	629
42.2.16	HAL_USART_Receive_DMA .....	630
42.2.17	HAL_USART_TransmitReceive_DMA .....	630
42.2.18	HAL_USART_DMAPause .....	630
42.2.19	HAL_USART_DMAResume .....	631
42.2.20	HAL_USART_DMAStop .....	631



42.2.21	HAL_USART_IRQHandler .....	631
42.2.22	HAL_USART_TxCpltCallback .....	631
42.2.23	HAL_USART_TxHalfCpltCallback.....	632
42.2.24	HAL_USART_RxCpltCallback.....	632
42.2.25	HAL_USART_RxHalfCpltCallback .....	632
42.2.26	HAL_USART_TxRxCpltCallback.....	632
42.2.27	HAL_USART_ErrorCallback .....	632
42.2.28	HAL_USART_GetState .....	633
42.2.29	HAL_USART_GetError.....	633
42.3	USART Firmware driver defines.....	633
42.3.1	USART.....	633
<b>43</b>	<b>HAL WWDG Generic Driver .....</b>	<b>641</b>
43.1	WWDG Firmware driver registers structures.....	641
43.1.1	WWDG_InitTypeDef .....	641
43.1.2	WWDG_HandleTypeDef .....	641
43.2	WWDG Firmware driver API description .....	641
43.2.1	WWDG specific features .....	641
43.2.2	How to use this driver .....	642
43.2.3	Initialization and de-initialization functions .....	642
43.2.4	IO operation functions .....	643
43.2.5	Peripheral State functions .....	643
43.2.6	HAL_WWDG_Init.....	643
43.2.7	HAL_WWDG_DeInit.....	643
43.2.8	HAL_WWDG_MspInit.....	644
43.2.9	HAL_WWDG_MspDeInit .....	644
43.2.10	HAL_WWDG_WakeupCallback .....	644
43.2.11	HAL_WWDG_Start.....	644
43.2.12	HAL_WWDG_Start_IT.....	645
43.2.13	HAL_WWDG_Refresh.....	645
43.2.14	HAL_WWDG_IRQHandler .....	645
43.2.15	HAL_WWDG_WakeupCallback .....	645
43.2.16	HAL_WWDG_GetState .....	646
43.3	WWDG Firmware driver defines.....	646
43.3.1	WWDG.....	646
<b>44</b>	<b>FAQs.....</b>	<b>650</b>
<b>45</b>	<b>Revision history .....</b>	<b>654</b>

## List of tables

Table 1: Acronyms and definitions.....	36
Table 2: HAL drivers files.....	38
Table 3: User-application files .....	39
Table 4: APIs classification .....	44
Table 5: List of devices supported by HAL drivers .....	45
Table 7: HAL API naming rules .....	51
Table 8: Macros handling interrupts and specific clock configurations .....	52
Table 9: Callback functions.....	53
Table 10: HAL generic APIs .....	54
Table 11: HAL extension APIs .....	55
Table 12: Define statements used for HAL configuration .....	59
Table 13: Description of GPIO_InitTypeDef structure .....	61
Table 14: Description of EXTI configuration macros .....	63
Table 15: MSP functions.....	68
Table 16: Timeout values .....	72
Table 17: IRDA frame formats .....	314
Table 18: Number of wait states (WS) vs SYSCLK frequency .....	395
Table 19: Smartcard frame formats.....	485
Table 20: Maximum SPI frequency for 8-bit SPI data transfers .....	505
Table 21: Maximum SPI frequency for 16-bit SPI data transfers .....	505
Table 22: UART frame formats.....	602
Table 23: USART frame formats .....	625
Table 24: Document revision history .....	654

## List of figures

Figure 1: Example of project template .....	41
Figure 2: Adding device-specific functions .....	56
Figure 3: Adding family-specific functions .....	56
Figure 4: Adding new peripherals .....	57
Figure 5: Updating existing APIs .....	57
Figure 6: File inclusion model .....	58
Figure 7: HAL driver model .....	66

# 1 Acronyms and definitions

Table 1: Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
ANSI	American National Standards Institute
API	Application Programming Interface
BSP	Board Support Package
CAN	Controller area network
CEC	Consumer electronic controller
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DMA	Direct Memory Access
ETH	Ethernet controller
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
HCD	USB Host Controller Driver
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LCD	Liquid Crystal Display Controller
MSP	MCU Specific Package
NAND	NAND Flash memory
NOR	Nor Flash memory
NVIC	Nested Vectored Interrupt Controller
PCD	USB Peripheral Controller Driver
PWR	Power controller
RCC	Reset and clock controller
RTC	Real-time clock
SD	Secure Digital
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface

Acronym	Definition
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

## 2 Overview of HAL drivers

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

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

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

The HAL main features are the following:

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

### 2.1 HAL and user-application files

#### 2.1.1 HAL driver files

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

**Table 2: HAL drivers files**

File	Description
<i>stm32f1xx_hal_ppp.c</i>	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. <i>Example: stm32f1xx_hal_adc.c, stm32f1xx_hal_irda.c, ...</i>
<i>stm32f1xx_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: stm32f1xx_hal_adc.h, stm32f1xx_hal_irda.h, ...</i>

File	Description
<i>stm32f1xx_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: stm32f1xx_hal_adc_ex.c, stm32f1xx_hal_dma_ex.c, ...</i>
<i>stm32f1xx_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: stm32f1xx_hal_adc_ex.h, stm32f1xx_hal_dma_ex.h, ...</i>
<i>stm32f1xx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
<i>stm32f1xx_hal.h</i>	stm32f1xx_hal.c header file
<i>stm32f1xx_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>stm32f1xx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application.
<i>stm32f1xx_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_stm32f1xx.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 : <ul style="list-style-type: none"> <li>relocate the vector table in internal SRAM.</li> </ul>
<i>startup_stm32f1xx.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>stm32f1xx_flash.icf</i> (optional)	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
<i>stm32f1xx_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>stm32f1xx_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>stm32f1xx_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>stm32f1xx_hal.c</i> ) used as HAL timebase. By default, this function is called each 1ms in SysTick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
<i>main.c/h</i>	This file contains the main program routine, mainly: <ul style="list-style-type: none"> <li>the call to HAL_Init()</li> <li>assert_failed() implementation</li> <li>system clock configuration</li> <li>peripheral HAL initialization and user application code.</li> </ul>

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

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

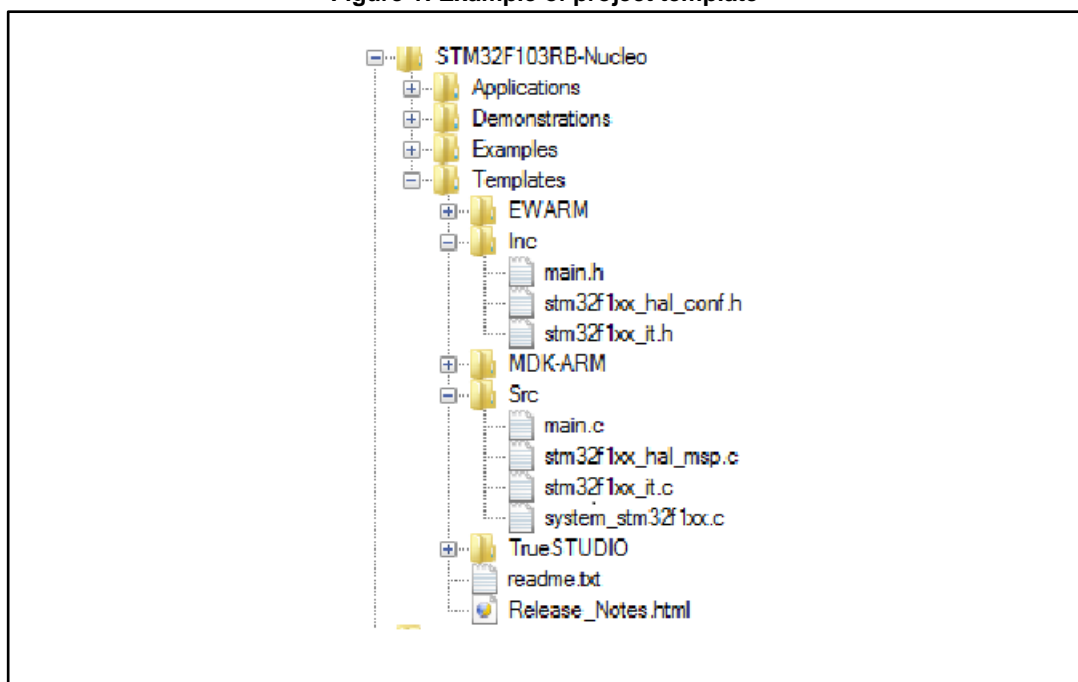
- It contains sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows to configure the CMSIS and HAL drivers accordingly.
- It provides 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,
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 (STM32F101xG) || defined (STM32F103x6) || defined (STM32F103xB) ||
defined (STM32F105xC) || defined (STM32F107xC) || defined (STM32F103xE) || defined
(STM32F103xG)
/* ADC multimode */ HAL_StatusTypeDef
HAL_ADCEX_MultiModeStart_DMA(ADC_HandleTypeDef* hadc, uint32_t *pData,
uint32_t Length);
HAL_StatusTypeDef HAL_ADCEX_MultiModeStop_DMA(ADC_HandleTypeDef* hadc);
#endif /* STM32F101xG || defined STM32F103x6 || defined STM32F103xB || defined
STM32F105xC || defined STM32F107xC || defined STM32F103xE || defined STM32F103xG */
```



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

**Notes:**

<sup>(1)</sup>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

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F1 00xB	STM32F1 00xE	STM32F1 01x6	STM32F1 01xB	STM32F1 01xE	STM32F1 01xG	STM32F1 02x6	STM32F1 02xB	STM32F1 03x6	STM32F1 03xB	STM32F1 03xE	STM32F1 03xG	STM32F1 05xC	STM32F1 07xC
stm32f1xx_hal.c stm32f1xx_hal.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_adc.c stm32f1xx_hal_adc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_adc_ex.c stm32f1xx_hal_adc_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_can.c stm32f1xx_hal_can.h	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_cec.c stm32f1xx_hal_cec.h	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No
stm32f1xx_hal_cortex.c stm32f1xx_hal_cortex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_crc.c stm32f1xx_hal_crc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_dac.c stm32f1xx_hal_dac.h	Yes	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes
stm32f1xx_hal_dac_ex.c stm32f1xx_hal_dac_ex.h	Yes	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F100xB	STM32F100xE	STM32F101x6	STM32F101xB	STM32F101xE	STM32F101xG	STM32F102x6	STM32F102xB	STM32F103x6	STM32F103xB	STM32F103xE	STM32F103xG	STM32F105xC	STM32F107xC
stm32f1xx_hal_dma. c stm32f1xx_hal_dma. h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_dma _ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_eth.c stm32f1xx_hal_eth.h	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes
stm32f1xx_hal_flash. c stm32f1xx_hal_flash. h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_flash _ex.c stm32f1xx_hal_flash _ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_gpio. c stm32f1xx_hal_gpio. h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_gpio _ex.c stm32f1xx_hal_gpio _ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_hcd.c stm32f1xx_hal_hcd. h	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F1 00xB	STM32F1 00xE	STM32F1 01x6	STM32F1 01xB	STM32F1 01xE	STM32F1 01xG	STM32F1 02x6	STM32F1 02xB	STM32F1 03x6	STM32F1 03xB	STM32F1 03xE	STM32F1 03xG	STM32F1 05xC	STM32F1 07xC
stm32f1xx_hal_i2c.c stm32f1xx_hal_i2c.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_i2s.c stm32f1xx_hal_i2s.h	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes
stm32f1xx_hal_irda. c stm32f1xx_hal_irda. h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_iwdg. c stm32f1xx_hal_iwdg. h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_msp _template.c	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stm32f1xx_hal_nand. c stm32f1xx_hal_nand. h	No	No	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_nor.c stm32f1xx_hal_nor.h	No	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_pccar. d.c stm32f1xx_hal_pccar. d.h	No	No	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_pcd.c stm32f1xx_hal_pcd. h	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F1 00xB	STM32F1 00xE	STM32F1 01x6	STM32F1 01xB	STM32F1 01xE	STM32F1 01xG	STM32F1 02x6	STM32F1 02xB	STM32F1 03x6	STM32F1 03xB	STM32F1 03xE	STM32F1 03xG	STM32F1 05xC	STM32F1 07xC
stm32f1xx_hal_pcd_ ex.c stm32f1xx_hal_pcd_ ex.h	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rcc.c stm32f1xx_hal_rcc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rcc_ ex.c stm32f1xx_hal_rcc_ ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rtc.c stm32f1xx_hal_rtc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rtc_e x.c stm32f1xx_hal_rtc_e x.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_sd.c stm32f1xx_hal_sd.h	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_smar tcard.c stm32f1xx_hal_smar tcard.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_spi.c stm32f1xx_hal_spi.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_spi_e x.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes



Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F100xB	STM32F100xE	STM32F101x6	STM32F101xB	STM32F101xE	STM32F101xG	STM32F102x6	STM32F102xB	STM32F103x6	STM32F103xB	STM32F103xE	STM32F103xG	STM32F105xC	STM32F107xC
stm32f1xx_hal_sram.c stm32f1xx_hal_sram.h	No	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_tim.c stm32f1xx_hal_tim.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_tim_ex.c stm32f1xx_hal_tim_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_uart.c stm32f1xx_hal_uart.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_usart.c stm32f1xx_hal_usart.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_wwdg.c stm32f1xx_hal_wwdg.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_fsmc.c stm32f1xx_ll_fsmc.h	No	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_ll_sdmmc.c stm32f1xx_ll_sdmmc.h	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No

## Overview of HAL drivers

UM1850

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F1 00xB	STM32F1 00xE	STM32F1 01x6	STM32F1 01xB	STM32F1 01xE	STM32F1 01xG	STM32F1 02x6	STM32F1 02xB	STM32F1 03x6	STM32F1 03xB	STM32F1 03xE	STM32F1 03xG	STM32F1 05xC	STM32F1 07xC
stm32f1xx_ll_usb.c stm32f1xx_ll_usb.h	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

## 2.5 HAL drivers rules

### 2.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

**Table 7: HAL API naming rules**

	Generic	Family specific	Device specific
<b>File names</b>	<i>stm32f1xx_hal_ppp (c/h)</i>	<i>stm32f1xx_hal_ppp_ex (c/h)</i>	<i>stm32f1xx_hal_ppp_ex (c/h)</i>
<b>Module name</b>	<i>HAL_PPP_MODULE</i>		
<b>Function name</b>	<i>HAL_PPP_Function</i> <i>HAL_PPP_FeatureFunction_MODE</i>	<i>HAL_PPPEX_Function</i> <i>HAL_PPPEX_FeatureFunction_MODE</i>	<i>HAL_PPPEX_Function</i> <i>HAL_PPPEX_FeatureFunction_MODE</i>
<b>Handle name</b>	<i>PPP_HandleTypeDef</i>	NA	NA
<b>Init structure name</b>	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
<b>Enum name</b>	<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 STM32F1xx reference manuals.
- Peripheral registers are declared in the *PPP\_TypeDef* structure (e.g. *ADC\_TypeDef*) in *stm32f1xxx.h* header file. *stm32f1xxx.h* corresponds to *stm32f100xb.h*, *stm32f100xe.h*, *stm32f101x6.h*, *stm32f101xb.h*, *stm32f101xe.h*, *stm32f101xg.h*, *stm32f102x6.h*, *stm32f102xb.h*, *stm32f103x6.h*, *stm32f103xb.h*, *stm32f103xe.h*, *stm32f103xg.h*, *stm32f105xc.h* and *stm32f107xc.h*.
- Peripheral function names are prefixed by *HAL\_*, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (e.g. *HAL\_UART\_Transmit()*). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named *PPP\_InitTypeDef* (e.g. *ADC\_InitTypeDef*).
- The structure containing the Specific configuration parameters for the PPP peripheral are named *PPP\_xxxxConfTypeDef* (e.g. *ADC\_ChannelConfTypeDef*).
- Peripheral handle structures are named *PPP\_HandleTypeDef* (e.g. *DMA\_HandleTypeDef*).
- The functions used to initialize the PPP peripheral according to parameters specified in *PPP\_InitTypeDef* are named *HAL\_PPP\_Init* (e.g. *HAL\_TIM\_Init()*).
- The functions used to reset the PPP peripheral registers to their default values are named *PPP\_DeInit*, e.g. *TIM\_DeInit*.

- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL\_PPP\_Function\_DMA ()*.
- The **Feature** prefix should refer to the new feature.  
Example: *HAL\_ADC\_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 8: 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 *stm32f1xx\_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 stm32f1xx\_it.c
- User callback functions.

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

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: HAL\_PPP\_MspInit() and HAL\_PPP\_MspDeInit
- Process complete callbacks : HAL\_PPP\_ProcessCpltCallback
- Error callback: HAL\_PPP\_ErrorCallback.

**Table 9: Callback functions**

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

## 2.6 HAL generic APIs

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

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

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

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

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

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

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

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

**Table 10: HAL generic APIs**

Function Group	Common API Name	Description
<i>Initialization group</i>	<code>HAL_ADC_Init()</code>	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	<code>HAL_ADC_DeInit()</code>	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>	<code>HAL_ADC_Start ()</code>	This function starts ADC conversions when the polling method is used
	<code>HAL_ADC_Stop ()</code>	This function stops ADC conversions when the polling method is used
	<code>HAL_ADC_PollForConversion()</code>	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.
	<code>HAL_ADC_Start_IT()</code>	This function starts ADC conversions when the interrupt method is used
	<code>HAL_ADC_Stop_IT()</code>	This function stops ADC conversions when the interrupt method is used
	<code>HAL_ADC_IRQHandler()</code>	This function handles ADC interrupt requests

Function Group	Common API Name	Description
	<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, *stm32f1xx\_hal\_ppp\_ex.c*, that includes all the specific functions and define statements (*stm32f1xx\_hal\_ppp\_ex.h*) for a given part number.

Below an example based on the ADC peripheral:

**Table 11: HAL extension APIs**

Function Group	Common API Name
<i>HAL_ADCEx_CalibrationStart()</i>	This function is used to start the automatic ADC calibration

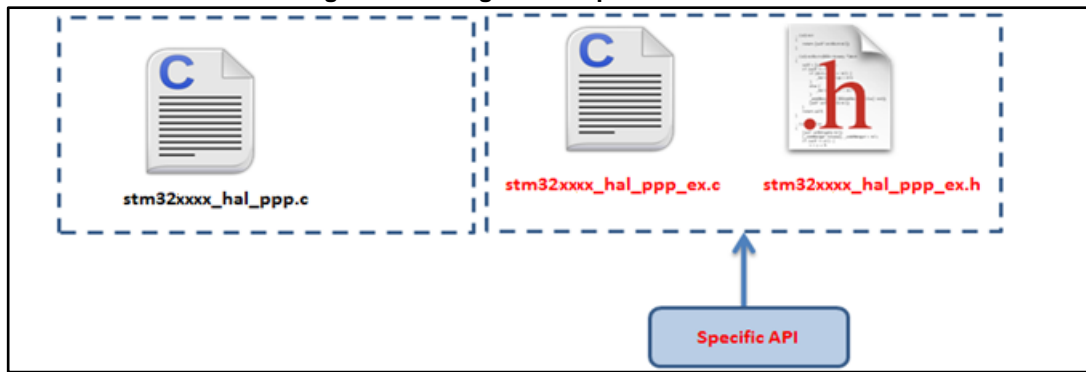
### 2.7.2 HAL extension model cases

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 *stm32f1xx\_hal\_ppp\_ex.c* extension file. They are named *HAL\_PPPEX\_Function()*.

Figure 2: Adding device-specific functions



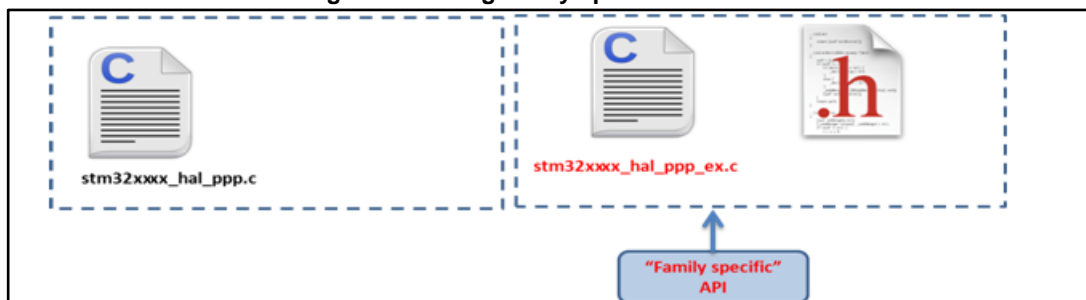
Example: `stm32f1xx_hal_adc_ex.c/h`

```
#if defined (STM32F101xG) || defined (STM32F103x6) || defined (STM32F103xB) ||
defined (STM32F105xC) ||
defined (STM32F107xC) || defined (STM32F103xE) || defined (STM32F103xG)
/* ADC multimode */
HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA(ADC_HandleTypeDef *hadc, uint32_t
*pData, uint32_t Length);
HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA(ADC_HandleTypeDef *hadc);
#endif /* STM32F101xG || defined STM32F103x6 || defined STM32F103xB || defined
STM32F105xC ||
defined STM32F107xC || defined STM32F103xE || defined STM32F103xG */
```

### 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



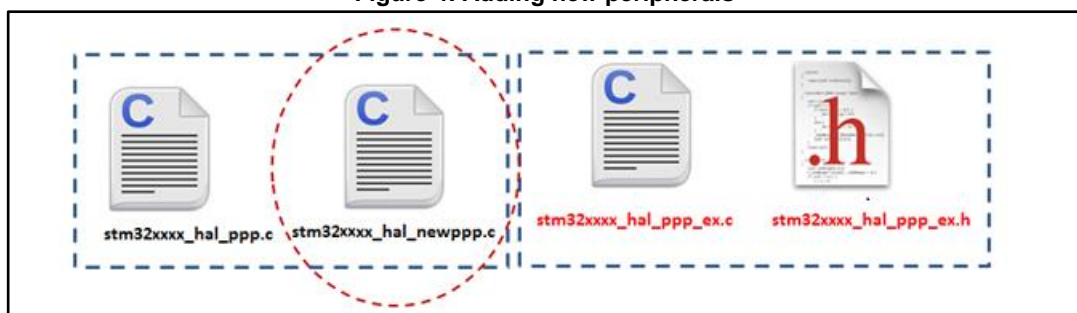
### Case3 : Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in `stm32f1xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32lxx_hal_conf.h` using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```



Figure 4: Adding new peripherals

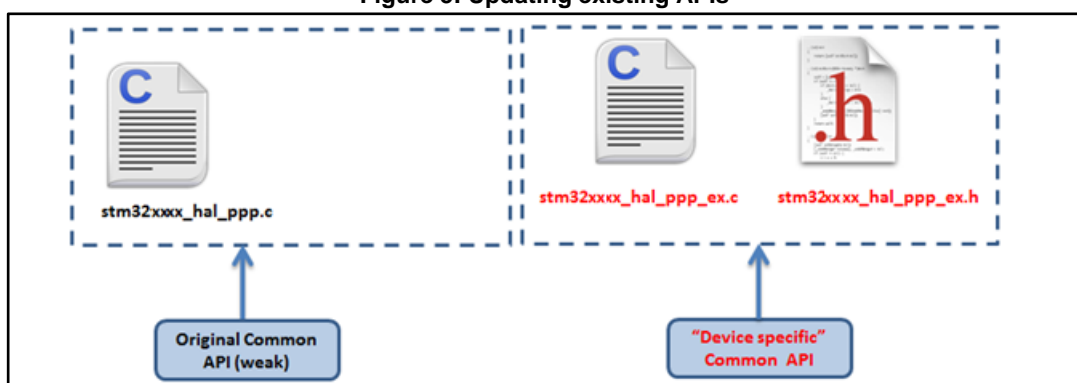


Example: stm32f1xx\_hal\_lcd.c/h

#### Case4: Updating existing common APIs

In this case, the routines are defined with the same names in the stm32f1xx\_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.

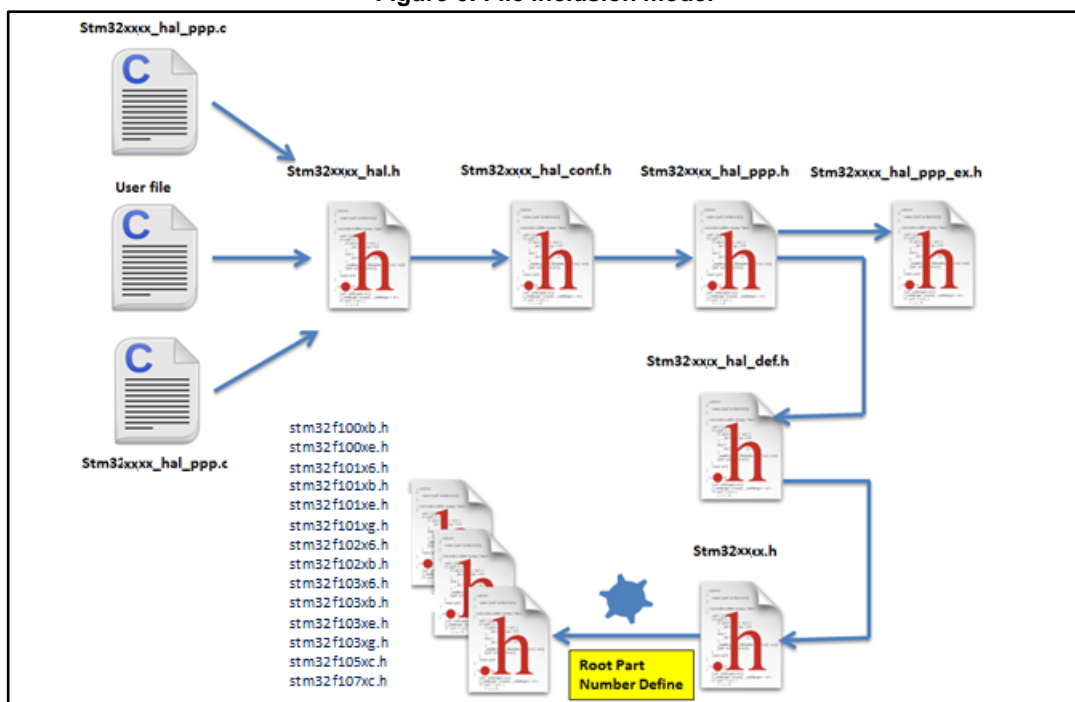
Example:

```
#if defined (STM32F100xB)
typedef struct
{
  (...)
}PPP_InitTypeDef;
#endif /* STM32F100xB */
```

## 2.8 File inclusion model

The header of the common HAL driver file (stm32f1xx\_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 stm32f1xx_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 *stm32f1xx\_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 `stm32f1xx_hal_def.h` file calls the `stm32f1xx.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 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, `stm32f1xx_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 12: Define statements used for HAL configuration**

Configuration item	Description	Default Value
<b>HSE_VALUE</b>	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 Hz on STM3210C-EVAL, otherwise 8 000 000 Hz
<b>HSE_STARTUP_TIMEOUT</b>	Timeout for HSE start up, expressed in ms	5000
<b>HSI_VALUE</b>	Defines the value of the internal oscillator (HSI) expressed in Hz.	8 000 000 Hz
<b>LSE_VALUE</b>	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 Hz
<b>LSE_STARTUP_TIMEOUT</b>	Timeout for LSE start up, expressed in ms	5000
<b>VDD_VALUE</b>	VDD value	3300 (mV)
<b>USE_RTOS</b>	Enables the use of RTOS	FALSE (for future use)
<b>PREFETCH_ENABLE</b>	Enables prefetch feature	TRUE



The `stm32f1xx_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 `stm32f1xx_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 `stm32f1xx_hal_rcc_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 `stm32f1xx_hal_rcc.h` and `stm32f1xx_hal_rcc_ex.h`. They allow 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 stm32f1xx\_it.c and implement HAL\_GPIO\_EXTI\_Callback()

The table below describes the GPIO\_InitTypeDef structure field.

**Table 13: 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"> <li><u>GPIO mode</u> <ul style="list-style-type: none"> <li>GPIO_MODE_INPUT : Input Floating</li> <li>GPIO_MODE_OUTPUT_PP : Output Push Pull</li> <li>GPIO_MODE_OUTPUT_OD : Output Open Drain</li> <li>GPIO_MODE_AF_PP : Alternate Function Push Pull</li> <li>GPIO_MODE_AF_OD : Alternate Function Open Drain</li> <li>GPIO_MODE_ANALOG : Analog mode</li> </ul> </li> <li><u>External Interrupt Mode</u> <ul style="list-style-type: none"> <li>GPIO_MODE_IT_RISING : Rising edge trigger detection</li> <li>GPIO_MODE_IT_FALLING : Falling edge trigger detection</li> <li>GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection</li> </ul> </li> <li><u>External Event Mode</u> <ul style="list-style-type: none"> <li>GPIO_MODE_EVT_RISING : Rising edge trigger detection</li> <li>GPIO_MODE_EVT_FALLING : Falling edge trigger detection</li> <li>GPIO_MODE_EVT_RISING_FALLING : Rising/Falling edge trigger detection</li> </ul> </li> </ul>
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: 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_HIGH

Please find below typical GPIO configuration examples:

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

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

- Configuring PA0 as external interrupt with falling edge sensitivity:

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

### 2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, stm32f1xx\_hal\_cortex.c, provides APIs to handle NVIC and SysTick. The supported APIs include:

- HAL\_NVIC\_SetPriority()/ HAL\_NVIC\_SetPriorityGrouping()
- HAL\_NVIC\_GetPriority() / HAL\_NVIC\_GetPriorityGrouping()
- HAL\_NVIC\_EnableIRQ()/HAL\_NVIC\_DisableIRQ()
- HAL\_NVIC\_SystemReset()
- HAL\_SYSTICK\_IRQHandler()
- HAL\_NVIC\_GetPendingIRQ() / HAL\_NVIC\_SetPendingIRQ () / HAL\_NVIC\_ClearPendingIRQ()
- HAL\_NVIC\_GetActive(IRQn)
- HAL\_SYSTICK\_Config()
- HAL\_SYSTICK\_CLKSourceConfig()
- HAL\_SYSTICK\_Callback()

### 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\_ConfigPVD()
  - HAL\_PWR\_EnablePVD() / HAL\_PWR\_DisablePVD()
  - HAL\_PWR\_PVD\_IRQHandler()
  - HAL\_PWR\_PVDCallback()
- Wakeup pin configuration
  - HAL\_PWR\_EnableWakeUpPin() / HAL\_PWR\_DisableWakeUpPin()
- Low power mode entry
  - HAL\_PWR\_EnterSLEEPMode()
  - HAL\_PWR\_EnterSTOPMode()
  - HAL\_PWR\_EnterSTANDBYMode()

### 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 Ethernet are configured within the HAL drivers of these peripheral through the macros given in the table below. The EXTI internal connections depend on the targeted STM32 microcontroller (refer to the product datasheet for more details):

Table 14: Description of EXTI configuration macros

Macros	Description
<code>__HAL_PPP_{SUBBLOCK}_EXTI_ENABLE_IT()</code>	Enables a given EXTI line interrupt Example: <code>__HAL_PWR_PVD_EXTI_ENABLE_IT()</code>
<code>__HAL_PPP_{SUBBLOCK}_EXTI_DISABLE_IT()</code>	Disables a given EXTI line. Example: <code>__HAL_PWR_PVD_EXTI_DISABLE_IT()</code>
<code>__HAL_PPP_{SUBBLOCK}_EXTI_GET_FLAG()</code>	Gets a given EXTI line interrupt flag pending bit status. Example: <code>__HAL_PWR_PVD_EXTI_GET_FLAG()</code>
<code>__HAL_PPP_{SUBBLOCK}_EXTI_CLEAR_FLAG()</code>	Clears a given EXTI line interrupt flag pending bit. Example; <code>__HAL_PWR_PVD_EXTI_CLEAR_FLAG()</code>
<code>__HAL_PPP_{SUBBLOCK}_EXTI_GENERATE_SWIT()</code>	Generates a software interrupt for a given EXTI line. Example: <code>__HAL_PWR_PVD_EXTI_GENERATE_SWIT()</code>
<code>__HAL_PPP_SUBBLOCK_EXTI_ENABLE_EVENT()</code>	Enable a given EXTI line event Example: <code>__HAL_RTC_WAKEUP_EXTI_ENABLE_EVENT()</code>
<code>__HAL_PPP_SUBBLOCK_EXTI_DISABLE_EVENT()</code>	Disable a given EXTI line event Example: <code>__HAL_RTC_WAKEUP_EXTI_DISABLE_EVENT()</code>
<code>__HAL_PPP_SUBBLOCK_EXTI_ENABLE_RISING_EDGE()</code>	Configure an EXTI Interrupt or Event on rising edge
<code>__HAL_PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()</code>	Enable an EXTI Interrupt or Event on Falling edge
<code>__HAL_PPP_SUBBLOCK_EXTI_DISABLE_RISING_EDGE()</code>	Disable an EXTI Interrupt or Event on rising edge
<code>__HAL_PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()</code>	Disable an EXTI Interrupt or Event on Falling edge
<code>__HAL_PPP_SUBBLOCK_EXTI_ENABLE_RISING_FALLING_EDGE()</code>	Enable an EXTI Interrupt or Event on Rising/Falling edge
<code>__HAL_PPP_SUBBLOCK_EXTI_DISABLE_RISING_FALLING_EDGE()</code>	Disable an EXTI Interrupt or Event on Rising/Falling edge

If the EXTI interrupt mode is selected, the user application must call `HAL_PPP_FUNCTION_IRQHandler()` (for example `HAL_PWR_PVD_IRQHandler()`), from `stm32f1xx_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 channel, `HAL_DMA_Init()` API allows programming the required configuration through the following parameters:

- Transfer Direction
- Source and Destination data formats
- Circular, Normal or peripheral flow control mode
- Channels Priority level
- Source and Destination Increment mode

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`: enable 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”).



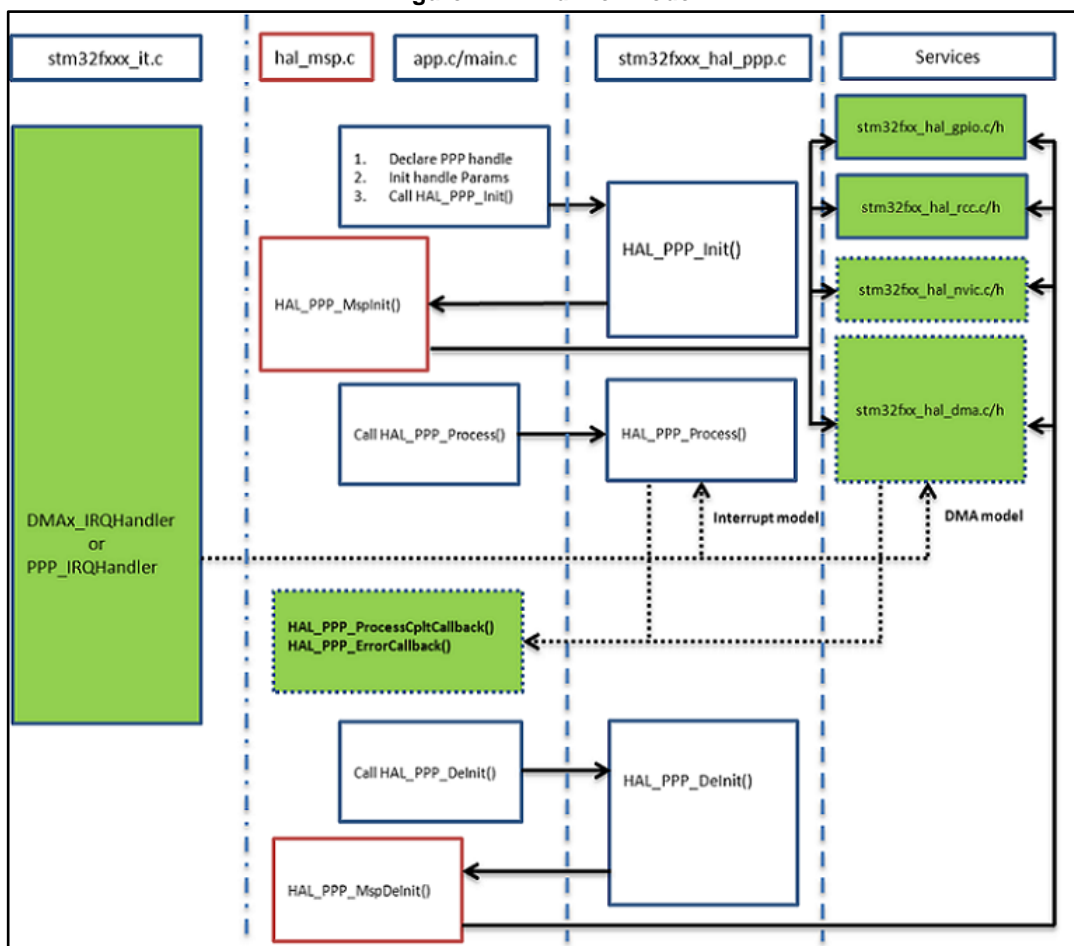
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

## 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 application user, the HAL driver and the interrupts.

Figure 7: HAL driver model



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

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

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

## 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 `stm32f1xx_hal.c`.

- `HAL_Init()`: this function must be called at application startup to
  - Initialize data/instruction cache and pre-fetch queue
  - Set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
  - Call `HAL_MspInit()` user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). `HAL_MspInit()` is defined as “weak” empty function in the HAL drivers.
- `HAL_DeInit()`
  - Resets all peripherals
  - Calls function `HAL_MspDeInit()` which is a user callback function to do system level De-Initializations.
- `HAL_GetTick()`: this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- `HAL_Delay()`: this function implements a delay (expressed in milliseconds) using the SysTick timer.

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

### 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 his own code. Please find below the typical Clock configuration sequence:

```
void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef clkinitstruct = {0};
  RCC_OscInitTypeDef oscinitstruct = {0};

  /* Configure PLLs -----*/
  /* PLL2 configuration: PLL2CLK = (HSE/HSEPrediv2Value)*PLL2MUL=(25/5)*8=40 MHz */
  /* PREDIV1 configuration: PREDIV1CLK = PLL2CLK / HSEPredivValue = 40 / 5 = 8 MHz */
  /* PLL configuration: PLLCLK = PREDIV1CLK * PLLMUL = 8 * 9 = 72 MHz */
  /* Enable HSE Oscillator and activate PLL with HSE as source */
  oscinitstruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  oscinitstruct.HSEState = RCC_HSE_ON;
  oscinitstruct.HSEPredivValue = RCC_HSE_PREDIV_DIV5;
  oscinitstruct.Prediv1Source = RCC_PREDIV1_SOURCE_PLL2;
  oscinitstruct.PLL.PLLState = RCC_PLL_ON;
  oscinitstruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  oscinitstruct.PLL.PLLMUL = RCC_PLL_MUL9;
  oscinitstruct.PLL2.PLL2State = RCC_PLL2_ON;
  oscinitstruct.PLL2.PLL2MUL = RCC_PLL2_MUL8;
  oscinitstruct.PLL2.HSEPrediv2Value = RCC_HSE_PREDIV2_DIV5;
  if (HAL_RCC_OscConfig(&oscinitstruct) != HAL_OK)
  { /* Initialization Error */
    while(1);
  }

  /* Select PLL as system clock source and configure the HCLK/PCLK1/PCLK2 clock
  dividers */
  clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
  RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
}
```

```

clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
clkinitstruct.APB2CLKDivider = RCC_HCLK_DIV1;
clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV2;
if (HAL_RCC_ClockConfig(&clkinitstruct, FLASH_LATENCY_2) != HAL_OK)
{ /* Initialization Error */
while(1);
}
}

```

### 2.12.2.3 HAL MSP initialization process

The peripheral initialization is done through *HAL\_PPP\_Init()* while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling MSP callback function *HAL\_PPP\_MspInit()*.

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

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

```

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

```

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

The HAL MSP callback is implemented inside the *stm32f1xx\_hal\_msp.c* file in the user folders. An *stm32f1xx\_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.

*stm32f1xx\_hal\_msp.c* file contains the following functions:

Table 15: MSP functions

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

By default, if no peripheral needs to be de-initialized during the program execution, the whole MSP initialization is done in *Hal\_MspInit()* and MSP De-Initialization in the *Hal\_MspDeInit()*. In this case the *HAL\_PPP\_MspInit()* and *HAL\_PPP\_MspDeInit()* are not implemented.

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

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

### 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_t tSize, uint32_t tTimeout)
{
if((pData == NULL) || (tSize == 0))
{
return HAL_ERROR;
}
(...) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(...)
return HAL_OK; }
```

#### 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 *stm32f1xx\_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)
{
}
```

*stm32f1xx\_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 *stm32f1xx\_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL\_PPP\_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
    PPP_TypeDef *Instance; /* Register base address */
}
```

```

PPP_InitTypeDef Init; /* PPP communication parameters */
HAL_StateTypeDef State; /* PPP communication state */
(...)
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;

```

The initialization is done as follows (UART example):

```

int main(void)
{
    /* Set User Parameters */
    UartHandle.Init.BaudRate = 9600;
    UartHandle.Init.WordLength = UART_DATABITS_8;
    UartHandle.Init.StopBits = UART_STOPBITS_1;
    UartHandle.Init.Parity = UART_PARITY_NONE;
    UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    UartHandle.Init.Mode = UART_MODE_TX_RX;
    UartHandle.Init.Instance = UART1;
    HAL_UART_Init(&UartHandle);
    (...)
}

void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
    static DMA_HandleTypeDef hdma_tx;
    static DMA_HandleTypeDef hdma_rx;
    (...)
    HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
    HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
    (...)
}

```

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

An example of use is illustrated below:

*main.c* file:

```

UART_HandleTypeDef UartHandle;
int main(void)
{
    /* Set User 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_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
    while (1);
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
}

void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
}

```

*stm32f1xx\_it.c* file:

```

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

```

*HAL\_USART\_TxCpltCallback()* and *HAL\_USART\_ErrorCallback()* should be linked in the *HAL\_PPP\_Process\_DMA()* function to the DMA transfer complete callback and the DMA transfer Error callback by using the following statement:

```

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

```

## 2.12.4 Timeout and error management

### 2.12.4.1 Timeout management

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

```

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

```

The timeout possible value are the following:

**Table 16: Timeout values**

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

**Notes:**

<sup>(1)</sup>HAL\_MAX\_DELAY is defined in the stm32f1xx\_hal\_def.h as 0xFFFFFFFF

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

Example of fixed timeout:

```

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

```

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

```

HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
    (...)
    timeout = HAL_GetTick() + Timeout;
    (...)
}

```



```

while(ProcessOngoing)
{
    (...)
    if(Timeout != HAL_MAX_DELAY)
    {
        if(HAL_GetTick() >= timeout)
        {
            /* Process unlocked */
            __HAL_UNLOCK(hppp);
            hppp->State= HAL_PPP_STATE_TIMEOUT;
            return hppp->State;
        }
    }
    (...)
}

```

### 2.12.4.2 Error management

The HAL drivers implement a check for the following items:

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

```

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

```

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

```

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

```

- Timeout error: the following statement is used when a timeout error occurs: while (Process ongoing)

```

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

```

When an error occurs during a peripheral process, *HAL\_PPP\_Process()* returns with a *HAL\_ERROR* status. The HAL PPP driver implements the *HAL\_PPP\_GetError()* to allow retrieving the origin of the error.

```

HAL_PPP_ErrorTypeDef HAL_PPP_GetError (PPP_HandleTypeDef *hppp);

```

In all peripheral handles, a *HAL\_PPP\_ErrorTypeDef* is defined and used to store the last error code.

```

typedef struct
{
    PPP_TypeDef * Instance; /* PPP registers base address */
    PPP_InitTypeDef Init; /* PPP initialization parameters */
    HAL_LockTypeDef Lock; /* PPP locking object */
    __IO HAL_PPP_StateTypeDef State; /* PPP state */
}

```

```

    IO HAL_PPP_ErrorTypeDef ErrorCode; /* PPP Error code */
    (...)
    /* PPP specific parameters */
}
PPP_HandleTypeDef;

```

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

```

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

```

**HAL\_PPP\_GetError ()** must be used in interrupt mode in the error callback:

```

void HAL_PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
    ErrorCode = HAL_PPP_GetError (hspi); /* retrieve 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 `stm32f1xx_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 `stm32f1xx_hal_conf.h`:

```

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

```

```
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr) ((void)0)
#endif /* USE_FULL_ASSERT */
```

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

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



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

## 3 HAL System Driver

### 3.1 HAL Firmware driver API description

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

#### 3.1.1 How to use this driver

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

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

#### 3.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

- 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\(\)](#)
- [HAL\\_InitTick\(\)](#)

#### 3.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond

- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier
- Get the device revision identifier
- Enable/Disable Debug module during Sleep mode
- Enable/Disable Debug module during STOP mode
- Enable/Disable Debug module during STANDBY mode
- [\*HAL\\_IncTick\(\)\*](#)
- [\*HAL\\_GetTick\(\)\*](#)
- [\*HAL\\_Delay\(\)\*](#)
- [\*HAL\\_SuspendTick\(\)\*](#)
- [\*HAL\\_ResumeTick\(\)\*](#)
- [\*HAL\\_GetHalVersion\(\)\*](#)
- [\*HAL\\_GetREVID\(\)\*](#)
- [\*HAL\\_GetDEVID\(\)\*](#)
- [\*HAL\\_DBGMCU\\_EnableDBGSleepMode\(\)\*](#)
- [\*HAL\\_DBGMCU\\_DisableDBGSleepMode\(\)\*](#)
- [\*HAL\\_DBGMCU\\_EnableDBGStopMode\(\)\*](#)
- [\*HAL\\_DBGMCU\\_DisableDBGStopMode\(\)\*](#)
- [\*HAL\\_DBGMCU\\_EnableDBGStandbyMode\(\)\*](#)
- [\*HAL\\_DBGMCU\\_DisableDBGStandbyMode\(\)\*](#)

### 3.1.4 HAL\_Init

Function Name	<b>HAL_StatusTypeDef HAL_Init (void )</b>
Function Description	This function configures the Flash prefetch, Configures time base source, NVIC and Low level hardware Note: This function is called at the beginning of program after reset and before the clock configuration Note: The time base configuration is based on MSI clock when exiting from Reset.
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 3.1.5 HAL\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_DeInit (void )</b>
Function Description	This function de-Initializes common part of the HAL and stops the source of time base.
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 3.1.6 HAL\_MspInit

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

### 3.1.7 HAL\_MspDeInit

Function Name	<b>void HAL_MspDeInit (void )</b>
Function Description	DeInitializes the MSP.
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 3.1.8 HAL\_InitTick

Function Name	<b>HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)</b>
Function Description	This function configures the source of the time base.
Parameters	<ul style="list-style-type: none"><li>• <b>TickPriority</b>: Tick interrupt priority.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 3.1.9 HAL\_IncTick

Function Name	<b>void HAL_IncTick (void )</b>
Function Description	This function is called to increment a global variable "uwTick" used as application time base.
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 3.1.10 HAL\_GetTick

Function Name	<b>uint32_t HAL_GetTick (void )</b>
Function Description	Provides a tick value in millisecond.
Return values	<ul style="list-style-type: none"><li>• tick value</li></ul>

### 3.1.11 HAL\_Delay

Function Name	<b>void HAL_Delay ( __IO uint32_t Delay)</b>
Function Description	This function provides accurate delay (in milliseconds) based on variable incremented.
Parameters	<ul style="list-style-type: none"><li>• <b>Delay</b>: specifies the delay time length, in milliseconds.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 3.1.12 HAL\_SuspendTick

Function Name	<b>void HAL_SuspendTick (void )</b>
Function Description	Suspend Tick increment.

Return values

- None

### 3.1.13 HAL\_ResumeTick

Function Name **void HAL\_ResumeTick (void )**

Function Description Resume Tick increment.

Return values

- None

### 3.1.14 HAL\_GetHalVersion

Function Name **uint32\_t HAL\_GetHalVersion (void )**

Function Description Returns the HAL revision.

Return values

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

### 3.1.15 HAL\_GetREVID

Function Name **uint32\_t HAL\_GetREVID (void )**

Function Description Returns the device revision identifier.

Return values

- Device revision identifier

### 3.1.16 HAL\_GetDEVID

Function Name **uint32\_t HAL\_GetDEVID (void )**

Function Description Returns the device identifier.

Return values

- Device identifier

### 3.1.17 HAL\_DBGMCU\_EnableDBGSleepMode

Function Name **void HAL\_DBGMCU\_EnableDBGSleepMode (void )**

Function Description Enable the Debug Module during SLEEP mode.

Return values

- None

### 3.1.18 HAL\_DBGMCU\_DisableDBGSleepMode

Function Name **void HAL\_DBGMCU\_DisableDBGSleepMode (void )**

Function Description Disable the Debug Module during SLEEP mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU\_IDCODE and DBGMCU\_CR are accessible only in

debug mode (not accessible by the user software in normal mode).

Return values

- None

### 3.1.19 HAL\_DBGMCU\_EnableDBGStopMode

Function Name **void HAL\_DBGMCU\_EnableDBGStopMode (void )**

Function Description Enable the Debug Module during STOP mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU\_IDCODE and DBGMCU\_CR are accessible only in debug mode (not accessible by the user software in normal mode).

Return values

- None

### 3.1.20 HAL\_DBGMCU\_DisableDBGStopMode

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

Function Description Disable the Debug Module during STOP mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU\_IDCODE and DBGMCU\_CR are accessible only in debug mode (not accessible by the user software in normal mode).

Return values

- None

### 3.1.21 HAL\_DBGMCU\_EnableDBGStandbyMode

Function Name **void HAL\_DBGMCU\_EnableDBGStandbyMode (void )**

Function Description Enable the Debug Module during STANDBY mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU\_IDCODE and DBGMCU\_CR are accessible only in debug mode (not accessible by the user software in normal mode).

Return values

- None

### 3.1.22 HAL\_DBGMCU\_DisableDBGStandbyMode

Function Name **void HAL\_DBGMCU\_DisableDBGStandbyMode (void )**

Function Description Disable the Debug Module during STANDBY mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU\_IDCODE and DBGMCU\_CR are accessible only in



debug mode (not accessible by the user software in normal mode).

Return values

- None

## 3.2 HAL Firmware driver defines

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

### 3.2.1 HAL

HAL

#### **HAL Private Constants**

\_\_STM32F1xx\_HAL\_VERSION\_MAIN [31:24] main version  
\_\_STM32F1xx\_HAL\_VERSION\_SUB1 [23:16] sub1 version  
\_\_STM32F1xx\_HAL\_VERSION\_SUB2 [15:8] sub2 version  
\_\_STM32F1xx\_HAL\_VERSION\_RC [7:0] release candidate  
\_\_STM32F1xx\_HAL\_VERSION  
IDCODE\_DEVID\_MASK

## 4 HAL ADC Generic Driver

### 4.1 ADC Firmware driver registers structures

#### 4.1.1 ADC\_InitTypeDef

**ADC\_InitTypeDef** is defined in the `stm32f1xx_hal_adc.h`

##### Data Fields

- **`uint32_t DataAlign`**
- **`uint32_t ScanConvMode`**
- **`uint32_t ContinuousConvMode`**
- **`uint32_t NbrOfConversion`**
- **`uint32_t DiscontinuousConvMode`**
- **`uint32_t NbrOfDiscConversion`**
- **`uint32_t ExternalTrigConv`**

##### Field Documentation

- **`uint32_t ADC_InitTypeDef::DataAlign`** Specifies ADC data alignment to right (MSB on register bit 11 and LSB on register bit 0) (default setting) or to left (if regular group: MSB on register bit 15 and LSB on register bit 4, if injected group (MSB kept as signed value due to potential negative value after offset application): MSB on register bit 14 and LSB on register bit 3). This parameter can be a value of [ADC\\_Data\\_align](#)
- **`uint32_t ADC_InitTypeDef::ScanConvMode`** Configures the sequencer of regular and injected groups. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by 'NbrOfConversion'/'InjectedNbrOfConversion' and each channel rank). Scan direction is upward: from rank1 to rank 'n'. This parameter can be a value of [ADC\\_Scan\\_mode](#) Note: For regular group, this parameter should be enabled in conversion either by polling (HAL\_ADC\_Start with Discontinuous mode and NbrOfDiscConversion=1) or by DMA (HAL\_ADC\_Start\_DMA), but not by interruption (HAL\_ADC\_Start\_IT): in scan mode, interruption is triggered only on the the last conversion of the sequence. All previous conversions would be overwritten by the last one. Injected group used with scan mode has not this constraint: each rank has its own result register, no data is overwritten.
- **`uint32_t ADC_InitTypeDef::ContinuousConvMode`** Specifies whether the conversion is performed in single mode (one conversion) or continuous mode for regular group, after the selected trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- **`uint32_t ADC_InitTypeDef::NbrOfConversion`** Specifies the number of ranks that will be converted within the regular group sequencer. To use regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min\_Data = 1 and Max\_Data = 16.
- **`uint32_t ADC_InitTypeDef::DiscontinuousConvMode`** Specifies whether the conversions sequence of regular group is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded.

Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE.

- ***uint32\_t ADC\_InitTypeDef::NbrOfDiscConversion*** Specifies the number of discontinuous conversions in which the main sequence of regular group (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min\_Data = 1 and Max\_Data = 8.
- ***uint32\_t ADC\_InitTypeDef::ExternalTrigConv*** Selects the external event used to trigger the conversion start of regular group. If set to ADC\_SOFTWARE\_START, external triggers are disabled. If set to external trigger source, triggering is on event rising edge. This parameter can be a value of [ADC\\_External\\_trigger\\_source\\_Regular](#)

#### 4.1.2 ADC\_ChannelConfTypeDef

***ADC\_ChannelConfTypeDef*** is defined in the stm32f1xx\_hal\_adc.h

##### Data Fields

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

##### Field Documentation

- ***uint32\_t ADC\_ChannelConfTypeDef::Channel*** Specifies the channel to configure into ADC regular group. This parameter can be a value of [ADC\\_channels](#) Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability. Note: On STM32F1 devices with several ADC: Only ADC1 can access internal measurement channels (VrefInt/TempSensor) Note: On STM32F10xx8 and STM32F10xxB devices: A low-amplitude voltage glitch may be generated (on ADC input 0) on the PA0 pin, when the ADC is converting with injection trigger. It is advised to distribute the analog channels so that Channel 0 is configured as an injected channel. Refer to errata sheet of these devices for more details.
- ***uint32\_t ADC\_ChannelConfTypeDef::Rank*** Specifies the rank in the regular group sequencer This parameter can be a value of [ADC\\_regular\\_rank](#) Note: In case of need to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions can be adjusted)
- ***uint32\_t ADC\_ChannelConfTypeDef::SamplingTime*** Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits). This parameter can be a value of [ADC\\_sampling\\_times](#) Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting. Note: In case of usage of internal measurement channels (VrefInt/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS\_vrefint, TS\_temp (values rough order: 5us to 17.1us min).

### 4.1.3 ADC\_AnalogWDGConfTypeDef

**ADC\_AnalogWDGConfTypeDef** is defined in the stm32f1xx\_hal\_adc.h

#### Data Fields

- *uint32\_t WatchdogMode*
- *uint32\_t Channel*
- *uint32\_t ITMode*
- *uint32\_t HighThreshold*
- *uint32\_t LowThreshold*
- *uint32\_t WatchdogNumber*

#### Field Documentation

- *uint32\_t ADC\_AnalogWDGConfTypeDef::WatchdogMode* Configures the ADC analog watchdog mode: single/all channels, regular/injected group. 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. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFF.
- *uint32\_t ADC\_AnalogWDGConfTypeDef::LowThreshold* Configures the ADC analog watchdog Low threshold value. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFF.
- *uint32\_t ADC\_AnalogWDGConfTypeDef::WatchdogNumber* Reserved for future use, can be set to 0

### 4.1.4 ADC\_HandleTypeDef

**ADC\_HandleTypeDef** is defined in the stm32f1xx\_hal\_adc.h

#### Data Fields

- *ADC\_TypeDef \* Instance*
- *ADC\_InitTypeDef Init*
- *\_\_IO uint32\_t NbrOfConversionRank*
- *DMA\_HandleTypeDef \* DMA\_Handle*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_ADC\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*

#### Field Documentation

- *ADC\_TypeDef\* ADC\_HandleTypeDef::Instance* Register base address
- *ADC\_InitTypeDef ADC\_HandleTypeDef::Init* ADC required parameters
- *\_\_IO uint32\_t ADC\_HandleTypeDef::NbrOfConversionRank* ADC conversion rank counter

- ***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 ADC Firmware driver API description

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

### 4.2.1 ADC peripheral features

- 12-bit resolution
- Interrupt generation at the end of regular conversion, end of injected conversion, and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for automatic conversion of channel 0 to channel 'n'.
- Data alignment with in-built data coherency.
- Channel-wise programmable sampling time.
- ADC conversion Regular or Injected groups.
- External trigger (timer or EXTI) with configurable polarity for both regular and injected groups.
- DMA request generation for transfer of conversions data of regular group.
- Multimode Dual mode (available on devices with 2 ADCs or more).
- Configurable DMA data storage in Multimode Dual mode (available on devices with 2 DCs or more).
- Configurable delay between conversions in Dual interleaved mode (available on devices with 2 DCs or more).
- ADC calibration
- ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

### 4.2.2 How to use this driver

#### Configuration of top level parameters related to ADC

1. Enable the ADC interface
  - As prerequisite, ADC clock must be configured at RCC top level.
  - One clock setting is mandatory: ADC clock (core and conversion clock):
    - Example: Into HAL\_ADC\_MspInit() (recommended code location) or with other device clock parameters configuration:
    - PeriphClkInit.PeriphClockSelection = RCC\_PERIPHCLK\_ADC;
    - PeriphClkInit.AdcClockSelection = RCC\_ADCPCLK2\_DIVx ;
    - HAL\_RCCEX\_PeriphCLKConfig(&PeriphClkInit);

2. ADC pins configuration
  - Enable the clock for the ADC GPIOs using macro `__HAL_RCC_GPIOx_CLK_ENABLE()`
  - Configure these ADC pins in analog mode using function `HAL_GPIO_Init()`
3. Optionally, in case of usage of ADC with interruptions:
  - Configure the NVIC for ADC using function `HAL_NVIC_EnableIRQ(ADCx_IRQn)`
  - Insert the ADC interruption handler function `HAL_ADC_IRQHandler()` into the function of corresponding ADC interruption vector `ADCx_IRQHandler()`.
4. Optionally, in case of usage of DMA:
  - Configure the DMA (DMA channel, mode normal or circular, ...) using function `HAL_DMA_Init()`.
  - Configure the NVIC for DMA using function `HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)`
  - Insert the ADC interruption handler function `HAL_ADC_IRQHandler()` into the function of corresponding DMA interruption vector `DMAx_Channelx_IRQHandler()`.

### Configuration of ADC, groups regular/injected, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ..., of regular group) using function `HAL_ADC_Init()`.
2. Configure the channels for regular group parameters (channel number, channel rank into sequencer, ..., into regular group) using function `HAL_ADC_ConfigChannel()`.
3. Optionally, configure the injected group parameters (conversion trigger, sequencer, ..., of injected group) and the channels for injected group parameters (channel number, channel rank into sequencer, ..., into injected group) using function `HAL_ADCEx_InjectedConfigChannel()`.
4. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function `HAL_ADC_AnalogWDGConfig()`.
5. Optionally, for devices with several ADC instances: configure the multimode parameters using function `HAL_ADCEx_MultiModeConfigChannel()`.

### Execution of ADC conversions

1. Optionally, perform an automatic ADC calibration to improve the conversion accuracy using function `HAL_ADCEx_Calibration_Start()`.
2. ADC driver can be used among three modes: polling, interruption, transfer by DMA.
  - ADC conversion by polling:
    - Activate the ADC peripheral and start conversions using function `HAL_ADC_Start()`
    - Wait for ADC conversion completion using function `HAL_ADC_PollForConversion()` (or for injected group: `HAL_ADCEx_InjectedPollForConversion()` )
    - Retrieve conversion results using function `HAL_ADC_GetValue()` (or for injected group: `HAL_ADCEx_InjectedGetValue()` )
    - Stop conversion and disable the ADC peripheral using function `HAL_ADC_Stop()`
  - ADC conversion by interruption:
    - Activate the ADC peripheral and start conversions using function `HAL_ADC_Start_IT()`

- Wait for ADC conversion completion by call of function HAL\_ADC\_ConvCpltCallback() (this function must be implemented in user program) (or for injected group: HAL\_ADCEX\_InjectedConvCpltCallback() )
- Retrieve conversion results using function HAL\_ADC\_GetValue() (or for injected group: HAL\_ADCEX\_InjectedGetValue() )
- Stop conversion and disable the ADC peripheral using function HAL\_ADC\_Stop\_IT()
- ADC conversion with transfer by DMA:
  - Activate the ADC peripheral and start conversions using function HAL\_ADC\_Start\_DMA()
  - Wait for ADC conversion completion by call of function HAL\_ADC\_ConvCpltCallback() or HAL\_ADC\_ConvHalfCpltCallback() (these functions must be implemented in user program)
  - Conversion results are automatically transferred by DMA into destination variable address.
  - Stop conversion and disable the ADC peripheral using function HAL\_ADC\_Stop\_DMA()
- For devices with several ADCs: ADC multimode conversion with transfer by DMA:
  - Activate the ADC peripheral (slave) and start conversions using function HAL\_ADC\_Start()
  - Activate the ADC peripheral (master) and start conversions using function HAL\_ADCEX\_MultiModeStart\_DMA()
  - Wait for ADC conversion completion by call of function HAL\_ADC\_ConvCpltCallback() or HAL\_ADC\_ConvHalfCpltCallback() (these functions must be implemented in user program)
  - Conversion results are automatically transferred by DMA into destination variable address.
  - Stop conversion and disable the ADC peripheral (master) using function HAL\_ADCEX\_MultiModeStop\_DMA()
  - Stop conversion and disable the ADC peripheral (slave) using function HAL\_ADC\_Stop\_IT()



Callback functions must be implemented in user program:

- HAL\_ADC\_ErrorCallback()
- HAL\_ADC\_LevelOutOfWindowCallback() (callback of analog watchdog)
- HAL\_ADC\_ConvCpltCallback()
- HAL\_ADC\_ConvHalfCpltCallback
- HAL\_ADCEX\_InjectedConvCpltCallback()

## Deinitialization of ADC

1. Disable the ADC interface
  - ADC clock can be hard reset and disabled at RCC top level.
  - Hard reset of ADC peripherals using macro \_\_ADCx\_FORCE\_RESET(), \_\_ADCx\_RELEASE\_RESET().
  - ADC clock disable using the equivalent macro/functions as configuration step.
    - Example: Into HAL\_ADC\_MspDeInit() (recommended code location) or with other device clock parameters configuration:
    - PeriphClkInit.PeriphClockSelection = RCC\_PERIPHCLK\_ADC
    - PeriphClkInit.AdcClockSelection = RCC\_ADCPLLCLK2\_OFF

- HAL\_RCCEx\_PeriphCLKConfig(&PeriphClkInit)
- 2. ADC pins configuration
  - Disable the clock for the ADC GPIOs using macro `__HAL_RCC_GPIOx_CLK_DISABLE()`
- 3. Optionally, in case of usage of ADC with interruptions:
  - Disable the NVIC for ADC using function `HAL_NVIC_EnableIRQ(ADCx_IRQn)`
- 4. Optionally, in case of usage of DMA:
  - Deinitialize the DMA using function `HAL_DMA_Init()`.
  - Disable the NVIC for DMA using function `HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)`

### 4.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the ADC.
- De-initialize the ADC.
- [\*HAL\\_ADC\\_Init\(\)\*](#)
- [\*HAL\\_ADC\\_DeInit\(\)\*](#)
- [\*HAL\\_ADC\\_MspInit\(\)\*](#)
- [\*HAL\\_ADC\\_MspDeInit\(\)\*](#)

### 4.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion of regular group.
- Stop conversion of regular group.
- Poll for conversion complete on regular group.
- Poll for conversion event.
- Get result of regular channel conversion.
- Start conversion of regular group and enable interruptions.
- Stop conversion of regular group and disable interruptions.
- Handle ADC interrupt request
- Start conversion of regular group and enable DMA transfer.
- Stop conversion of regular group and disable ADC DMA transfer.
- [\*HAL\\_ADC\\_Start\(\)\*](#)
- [\*HAL\\_ADC\\_Stop\(\)\*](#)
- [\*HAL\\_ADC\\_PollForConversion\(\)\*](#)
- [\*HAL\\_ADC\\_PollForEvent\(\)\*](#)
- [\*HAL\\_ADC\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_ADC\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_ADC\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_ADC\\_Stop\\_DMA\(\)\*](#)
- [\*HAL\\_ADC\\_GetValue\(\)\*](#)
- [\*HAL\\_ADC\\_IRQHandler\(\)\*](#)
- [\*HAL\\_ADC\\_ConvCpltCallback\(\)\*](#)
- [\*HAL\\_ADC\\_ConvHalfCpltCallback\(\)\*](#)
- [\*HAL\\_ADC\\_LevelOutOfWindowCallback\(\)\*](#)
- [\*HAL\\_ADC\\_ErrorCallback\(\)\*](#)



## 4.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog
- [HAL\\_ADC\\_ConfigChannel\(\)](#)
- [HAL\\_ADC\\_AnalogWDGConfig\(\)](#)

## 4.2.6 Peripheral State and Errors functions

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

- Check the ADC state
- Check the ADC error code
- [HAL\\_ADC\\_GetState\(\)](#)
- [HAL\\_ADC\\_GetError\(\)](#)

## 4.2.7 HAL\_ADC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)</b>
Function Description	Initializes the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• As prerequisite, ADC clock must be configured at RCC top level (clock source APB2). See commented example code below that can be copied and uncommented into HAL_ADC_MspInit().</li> <li>• Possibility to update parameters on the fly: This function initializes the ADC MSP (HAL_ADC_MspInit()) only when coming from ADC state reset. Following calls to this function can be used to reconfigure some parameters of ADC_InitTypeDef structure on the fly, without modifying MSP configuration. If ADC MSP has to be modified again, HAL_ADC_DeInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef".</li> <li>• This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef".</li> </ul>

## 4.2.8 HAL\_ADC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)</b>
---------------	--

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

Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b>: ADC handle</li></ul>
------------	---

Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>
---------------	--

#### 4.2.9 HAL\_ADC\_MspInit

Function Name	<b>void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)</b>
---------------	--

Function Description	Initializes the ADC MSP.
----------------------	--------------------------

Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b>: ADC handle</li></ul>
------------	---

Return values	<ul style="list-style-type: none"><li>• None</li></ul>
---------------	--

#### 4.2.10 HAL\_ADC\_MspDeInit

Function Name	<b>void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)</b>
---------------	--

Function Description	DeInitializes the ADC MSP.
----------------------	----------------------------

Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b>: ADC handle</li></ul>
------------	---

Return values	<ul style="list-style-type: none"><li>• None</li></ul>
---------------	--

#### 4.2.11 HAL\_ADC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)</b>
---------------	---

Function Description	Enables ADC, starts conversion of regular group.
----------------------	--

Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b>: ADC handle</li></ul>
------------	---

Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>
---------------	--

#### 4.2.12 HAL\_ADC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)</b>
---------------	--

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

Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b>: ADC handle</li></ul>
------------	---

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

Notes	<ul style="list-style-type: none"><li>• : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.</li></ul>
-------	---

**4.2.13 HAL\_ADC\_PollForConversion**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_PollForConversion</b> (ADC_HandleTypeDef * hadc, uint32_t Timeout)
Function Description	Wait for regular group conversion to be completed.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> <li>• <b>Timeout</b>: Timeout value in millisecond.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**4.2.14 HAL\_ADC\_PollForEvent**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_PollForEvent</b> (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)
Function Description	Poll for conversion event.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> <li>• <b>EventType</b>: the ADC event type. This parameter can be one of the following values: ADC_AWD_EVENT: ADC Analog watchdog event.</li> <li>• <b>Timeout</b>: Timeout value in millisecond.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**4.2.15 HAL\_ADC\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start_IT</b> (ADC_HandleTypeDef * hadc)
Function Description	Enables ADC, starts conversion of regular group with interruption.

**4.2.16 HAL\_ADC\_Stop\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop_IT</b> (ADC_HandleTypeDef * hadc)
Function Description	Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interruption of end-of-conversion, disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**4.2.17 HAL\_ADC\_Start\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start_DMA</b> (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t
---------------	---

**Length)**

Function Description Enables ADC, starts conversion of regular group and transfers result through DMA.

**4.2.18 HAL\_ADC\_Stop\_DMA**

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

Function Description Stop ADC conversion of regular group (and injected group in case of auto\_injection mode), disable ADC DMA transfer, disable ADC peripheral.

Parameters • **hadc**: ADC handle

Return values • HAL status.

Notes • : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL\_ADCEX\_InjectedStop function.  
 • For devices with several ADCs: This function is for single-ADC mode only. For multimode, use the dedicated MultimodeStop function.  
 • On STM32F1 devices, only ADC1 and ADC3 (ADC availability depending on devices) have DMA capability.

**4.2.19 HAL\_ADC\_GetValue**

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

Function Description Get ADC regular group conversion result.

Parameters • **hadc**: ADC handle

Return values • Converted value

Notes • Reading DR register automatically clears EOC (end of conversion of regular group) flag.

**4.2.20 HAL\_ADC\_IRQHandler**

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

Function Description Handles ADC interrupt request.

Parameters • **hadc**: ADC handle

Return values • None

**4.2.21 HAL\_ADC\_ConvCpltCallback**

Function Name **void HAL\_ADC\_ConvCpltCallback (ADC\_HandleTypeDef \***

**hadc)**

Function Description	Conversion complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 4.2.22 HAL\_ADC\_ConvHalfCpltCallback

Function Name	<b>void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)</b>
Function Description	Conversion DMA half-transfer callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 4.2.23 HAL\_ADC\_LevelOutOfWindowCallback

Function Name	<b>void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)</b>
Function Description	Analog watchdog callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 4.2.24 HAL\_ADC\_ErrorCallback

Function Name	<b>void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)</b>
Function Description	ADC error callback in non blocking mode (ADC conversion with interruption or transfer by DMA)
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 4.2.25 HAL\_ADC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)</b>
Function Description	Configures the the selected channel to be linked to the regular group.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> <li>• <b>sConfig</b>: Structure of ADC channel for regular group.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## Notes

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

#### 4.2.26 HAL\_ADC\_AnalogWDGConfig

Function Name	<b>HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)</b>
Function Description	Configures the analog watchdog.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> <li>• <b>AnalogWDGConfig</b>: Structure of ADC analog watchdog configuration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 4.2.27 HAL\_ADC\_GetState

Function Name	<b>HAL_ADC_StateTypeDef HAL_ADC_GetState (ADC_HandleTypeDef * hadc)</b>
Function Description	return the ADC state
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

#### 4.2.28 HAL\_ADC\_GetError

Function Name	<b>uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)</b>
Function Description	Return the ADC error code.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• ADC Error Code</li> </ul>

### 4.3 ADC Firmware driver defines

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

#### 4.3.1 ADC

ADC

*ADC analog watchdog mode*

ADC\_ANALOGWATCHDOG\_NONE  
ADC\_ANALOGWATCHDOG\_SINGLE\_REG  
ADC\_ANALOGWATCHDOG\_SINGLE\_INJEC  
ADC\_ANALOGWATCHDOG\_SINGLE\_REGINJEC  
ADC\_ANALOGWATCHDOG\_ALL\_REG  
ADC\_ANALOGWATCHDOG\_ALL\_INJEC  
ADC\_ANALOGWATCHDOG\_ALL\_REGINJEC

**ADC channels**

ADC\_CHANNEL\_0  
ADC\_CHANNEL\_1  
ADC\_CHANNEL\_2  
ADC\_CHANNEL\_3  
ADC\_CHANNEL\_4  
ADC\_CHANNEL\_5  
ADC\_CHANNEL\_6  
ADC\_CHANNEL\_7  
ADC\_CHANNEL\_8  
ADC\_CHANNEL\_9  
ADC\_CHANNEL\_10  
ADC\_CHANNEL\_11  
ADC\_CHANNEL\_12  
ADC\_CHANNEL\_13  
ADC\_CHANNEL\_14  
ADC\_CHANNEL\_15  
ADC\_CHANNEL\_16  
ADC\_CHANNEL\_17  
ADC\_CHANNEL\_TEMPSENSOR  
ADC\_CHANNEL\_VREFINT

**ADC conversion cycles**

ADC\_CONVERSIONCLOCKCYCLES\_SAMPLETIME\_1CYCLE5  
ADC\_CONVERSIONCLOCKCYCLES\_SAMPLETIME\_7CYCLES5  
ADC\_CONVERSIONCLOCKCYCLES\_SAMPLETIME\_13CYCLES5  
ADC\_CONVERSIONCLOCKCYCLES\_SAMPLETIME\_28CYCLES5  
ADC\_CONVERSIONCLOCKCYCLES\_SAMPLETIME\_41CYCLES5  
ADC\_CONVERSIONCLOCKCYCLES\_SAMPLETIME\_55CYCLES5  
ADC\_CONVERSIONCLOCKCYCLES\_SAMPLETIME\_71CYCLES5

ADC\_CONVERSIONCLOCKCYCLES\_SAMPLETIME\_239CYCLES5

#### **ADC conversion group**

ADC\_REGULAR\_GROUP

ADC\_INJECTED\_GROUP

ADC\_REGULAR\_INJECTED\_GROUP

#### **ADC data alignment**

ADC\_DATAALIGN\_RIGHT

ADC\_DATAALIGN\_LEFT

#### **ADC Error Code**

HAL\_ADC\_ERROR\_NONE      No error

HAL\_ADC\_ERROR\_INTERNAL    ADC IP internal error: if problem of clocking, enable/disable, erroneous state

HAL\_ADC\_ERROR\_OVR      Overrun error

HAL\_ADC\_ERROR\_DMA      DMA transfer error

#### **ADC Event type**

ADC\_AWD\_EVENT      ADC Analog watchdog event

ADC\_AWD1\_EVENT    ADC Analog watchdog 1 event: Alternate naming for compatibility with other STM32 devices having several analog watchdogs

#### **ADC Exported Macros**

\_\_HAL\_ADC\_ENABLE

##### **Description:**

- Enable the ADC peripheral.

##### **Parameters:**

- \_\_HANDLE\_\_: ADC handle

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

- None:

\_\_HAL\_ADC\_DISABLE

##### **Description:**

- Disable the ADC peripheral.

##### **Parameters:**

- \_\_HANDLE\_\_: ADC handle

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

- None:

\_\_HAL\_ADC\_ENABLE\_IT

##### **Description:**

- Enable the ADC end of conversion interrupt.

##### **Parameters:**

- \_\_HANDLE\_\_: ADC handle
- \_\_INTERRUPT\_\_: ADC Interrupt This parameter can be any combination of the



### \_\_HAL\_ADC\_DISABLE\_IT

following values:

- ADC\_IT\_EOC: ADC End of Regular Conversion interrupt source
- ADC\_IT\_JEOC: ADC End of Injected Conversion interrupt source
- ADC\_IT\_AWD: ADC Analog watchdog interrupt source

#### Return value:

- None:

#### Description:

- Disable the ADC end of conversion interrupt.

#### Parameters:

- \_\_HANDLE\_\_: ADC handle
- \_\_INTERRUPT\_\_: ADC Interrupt This parameter can be any combination of the following values:
  - ADC\_IT\_EOC: ADC End of Regular Conversion interrupt source
  - ADC\_IT\_JEOC: ADC End of Injected Conversion interrupt source
  - ADC\_IT\_AWD: ADC Analog watchdog interrupt source

#### Return value:

- None:

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

#### Description:

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

#### Parameters:

- \_\_HANDLE\_\_: ADC handle
- \_\_INTERRUPT\_\_: ADC interrupt source to check This parameter can be any combination of the following values:
  - ADC\_IT\_EOC: ADC End of Regular Conversion interrupt source
  - ADC\_IT\_JEOC: ADC End of Injected Conversion interrupt source
  - ADC\_IT\_AWD: ADC Analog watchdog interrupt source

#### Return value:

- None:

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

#### Description:

- Get the selected ADC's flag status.

#### Parameters:

- \_\_HANDLE\_\_: ADC handle

- `__FLAG__`: ADC flag This parameter can be any combination of the following values:
  - `ADC_FLAG_STRT`: ADC Regular group start flag
  - `ADC_FLAG_JSTRT`: ADC Injected group start flag
  - `ADC_FLAG_EOC`: ADC End of Regular conversion flag
  - `ADC_FLAG_JEOC`: ADC End of Injected conversion flag
  - `ADC_FLAG_AWD`: ADC Analog watchdog flag

**Return value:**

- None:

**Description:**

- Clear the ADC's pending flags.

**Parameters:**

- `__HANDLE__`: ADC handle
- `__FLAG__`: ADC flag This parameter can be any combination of the following values:
  - `ADC_FLAG_STRT`: ADC Regular group start flag
  - `ADC_FLAG_JSTRT`: ADC Injected group start flag
  - `ADC_FLAG_EOC`: ADC End of Regular conversion flag
  - `ADC_FLAG_JEOC`: ADC End of Injected conversion flag
  - `ADC_FLAG_AWD`: ADC Analog watchdog flag

**Return value:**

- None:

**Description:**

- Reset ADC handle state.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None:

`__HAL_ADC_CLEAR_FLAG``__HAL_ADC_RESET_HANDLE_STATE`**ADC external trigger enable for regular group**`ADC_EXTERNALTRIGCONVEDGE_NONE``ADC_EXTERNALTRIGCONVEDGE_RISING`**ADC External trigger selection for regular group**`ADC_EXTERNALTRIGCONV_T1_CC1`

< List of external triggers with generic trigger name, independently of

ADC\_EXTERNALTRIGCONV\_T1\_CC2  
 ADC\_EXTERNALTRIGCONV\_T2\_CC2  
 ADC\_EXTERNALTRIGCONV\_T3\_TRGO  
 ADC\_EXTERNALTRIGCONV\_T4\_CC4  
 ADC\_EXTERNALTRIGCONV\_EXT\_IT11    External triggers of regular group for ADC3 only  
  
 ADC\_EXTERNALTRIGCONV\_T2\_CC3  
 ADC\_EXTERNALTRIGCONV\_T3\_CC1  
 ADC\_EXTERNALTRIGCONV\_T5\_CC1  
 ADC\_EXTERNALTRIGCONV\_T5\_CC3  
 ADC\_EXTERNALTRIGCONV\_T8\_CC1  
 ADC\_EXTERNALTRIGCONV\_T1\_CC3    < External triggers of regular group for all ADC instances Note: TIM8\_TRGO is available on ADC1 and ADC2 only in high-density and  
  
 ADC\_EXTERNALTRIGCONV\_T8\_TRGO  
 ADC\_SOFTWARE\_START

**ADC flags definition**

ADC\_FLAG\_STRT    ADC Regular group start flag  
 ADC\_FLAG\_JSTRT    ADC Injected group start flag  
 ADC\_FLAG\_EOC    ADC End of Regular conversion flag  
 ADC\_FLAG\_JEOC    ADC End of Injected conversion flag  
 ADC\_FLAG\_AWD    ADC Analog watchdog flag

**ADC interrupts definition**

ADC\_IT\_EOC    ADC End of Regular Conversion interrupt source  
 ADC\_IT\_JEOC    ADC End of Injected Conversion interrupt source  
 ADC\_IT\_AWD    ADC Analog watchdog interrupt source

**ADC Private Constants**

ADC\_ENABLE\_TIMEOUT  
 ADC\_DISABLE\_TIMEOUT  
 ADC\_STAB\_DELAY\_US  
 ADC\_TEMPSENSOR\_DELAY\_US  
 ADC\_FLAG\_POSTCONV\_ALL

**ADC Private Macros**

ADC\_IS\_ENABLE

**Description:**

- Verification of ADC state: enabled or disabled.

**Parameters:**

- `__HANDLE__`: ADC handle

**ADC\_IS\_SOFTWARE\_START\_REGULAR****Return value:**

- SET: (ADC enabled) or RESET (ADC disabled)

**Description:**

- Test if conversion trigger of regular group is software start or external trigger.

**Parameters:**

- \_\_HANDLE\_\_: ADC handle

**Return value:**

- SET: (software start) or RESET (external trigger)

**ADC\_IS\_SOFTWARE\_START\_INJECTED****Description:**

- Test if conversion trigger of injected group is software start or external trigger.

**Parameters:**

- \_\_HANDLE\_\_: ADC handle

**Return value:**

- SET: (software start) or RESET (external trigger)

**ADC\_CLEAR\_ERRORCODE****Description:**

- Clear ADC error code (set it to error code: "no error")

**Parameters:**

- \_\_HANDLE\_\_: ADC handle

**Return value:**

- None:

**ADC\_SQR1\_L\_SHIFT****Description:**

- Set ADC number of conversions into regular channel sequence length.

**Parameters:**

- \_NbrOfConversion\_: Regular channel sequence length

**Return value:**

- None:

**ADC\_SMPR1****Description:**

- Set the ADC's sample time for channel numbers between 10 and 18.

**Parameters:**

ADC\_SMPR2

- `_SAMPLETIME_`: Sample time parameter.
- `_CHANNELNB_`: Channel number.

**Return value:**

- None:

**Description:**

- Set the ADC's sample time for channel numbers between 0 and 9.

**Parameters:**

- `_SAMPLETIME_`: Sample time parameter.
- `_CHANNELNB_`: Channel number.

**Return value:**

- None:

ADC\_SQR3\_RK

**Description:**

- Set the selected regular channel rank for rank between 1 and 6.

**Parameters:**

- `_CHANNELNB_`: Channel number.
- `_RANKNB_`: Rank number.

**Return value:**

- None:

ADC\_SQR2\_RK

**Description:**

- Set the selected regular channel rank for rank between 7 and 12.

**Parameters:**

- `_CHANNELNB_`: Channel number.
- `_RANKNB_`: Rank number.

**Return value:**

- None:

ADC\_SQR1\_RK

**Description:**

- Set the selected regular channel rank for rank between 13 and 16.

**Parameters:**

- `_CHANNELNB_`: Channel number.
- `_RANKNB_`: Rank number.

**Return value:**

- None:

ADC\_JSQR\_JL\_SHIFT

**Description:**

## ADC\_JSQR\_RK\_JL

- Set the injected sequence length.

**Parameters:**

- `_JSQR_JL_`: Sequence length.

**Return value:**

- None:

**Description:**

- Set the selected injected channel rank  
Note: on STM32F1 devices, channel rank position in JSQR register is depending on total number of ranks selected into injected sequencer (ranks sequence starting from 4-JL)

**Parameters:**

- `_CHANNELNB_`: Channel number.
- `_RANKNB_`: Rank number.
- `_JSQR_JL_`: Sequence length.

**Return value:**

- None:

**Description:**

- Enable ADC continuous conversion mode.

**Parameters:**

- `_CONTINUOUS_MODE_`: Continuous mode.

**Return value:**

- None:

## ADC\_CR2\_CONTINUOUS

## ADC\_CR1\_DISCONTINUOUS\_NUM

**Description:**

- Configures the number of discontinuous conversions for the regular group channels.

**Parameters:**

- `_NBR_DISCONTINUOUS_CONV_`:  
Number of discontinuous conversions.

**Return value:**

- None:

## ADC\_CR1\_SCAN\_SET

**Description:**

- Enable ADC scan mode to convert multiple ranks with sequencer.

**Parameters:**

- `_SCAN_MODE_`: Scan conversion mode.

ADC\_CONVCYCLES\_MAX\_RANGE

**Return value:**

- None:

**Description:**

- Get the maximum ADC conversion cycles on all channels.

**Parameters:**

- \_\_HANDLE\_\_: ADC handle

**Return value:**

- ADC: conversion cycles on all channels

IS\_ADC\_DATA\_ALIGN

IS\_ADC\_SCAN\_MODE

IS\_ADC\_EXTTRIG\_EDGE

IS\_ADC\_CHANNEL

IS\_ADC\_SAMPLE\_TIME

IS\_ADC\_REGULAR\_RANK

IS\_ADC\_ANALOG\_WATCHDOG\_MODE

IS\_ADC\_CONVERSION\_GROUP

IS\_ADC\_EVENT\_TYPE

**ADC range verification**

IS\_ADC\_RANGE

**ADC regular discontinuous mode number verification**

IS\_ADC\_REGULAR\_DISCONT\_NUMBER

**ADC regular nb conv verification**

IS\_ADC\_REGULAR\_NB\_CONV

**ADC rank into regular group**

ADC\_REGULAR\_RANK\_1

ADC\_REGULAR\_RANK\_2

ADC\_REGULAR\_RANK\_3

ADC\_REGULAR\_RANK\_4

ADC\_REGULAR\_RANK\_5

ADC\_REGULAR\_RANK\_6

ADC\_REGULAR\_RANK\_7

ADC\_REGULAR\_RANK\_8

ADC\_REGULAR\_RANK\_9

ADC\_REGULAR\_RANK\_10

ADC\_REGULAR\_RANK\_11

ADC\_REGULAR\_RANK\_12

ADC\_REGULAR\_RANK\_13

ADC\_REGULAR\_RANK\_14

ADC\_REGULAR\_RANK\_15

ADC\_REGULAR\_RANK\_16

**ADC sampling times**

ADC\_SAMPLETIME\_1CYCLE\_5      Sampling time 1.5 ADC clock cycle

ADC\_SAMPLETIME\_7CYCLES\_5      Sampling time 7.5 ADC clock cycles

ADC\_SAMPLETIME\_13CYCLES\_5      Sampling time 13.5 ADC clock cycles

ADC\_SAMPLETIME\_28CYCLES\_5      Sampling time 28.5 ADC clock cycles

ADC\_SAMPLETIME\_41CYCLES\_5      Sampling time 41.5 ADC clock cycles

ADC\_SAMPLETIME\_55CYCLES\_5      Sampling time 55.5 ADC clock cycles

ADC\_SAMPLETIME\_71CYCLES\_5      Sampling time 71.5 ADC clock cycles

ADC\_SAMPLETIME\_239CYCLES\_5      Sampling time 239.5 ADC clock cycles

**ADC sampling times all channels**

ADC\_SAMPLETIME\_ALLCHANNELS\_SMPR2BIT2

ADC\_SAMPLETIME\_ALLCHANNELS\_SMPR1BIT2

ADC\_SAMPLETIME\_ALLCHANNELS\_SMPR2BIT1

ADC\_SAMPLETIME\_ALLCHANNELS\_SMPR1BIT1

ADC\_SAMPLETIME\_ALLCHANNELS\_SMPR2BIT0

ADC\_SAMPLETIME\_ALLCHANNELS\_SMPR1BIT0

ADC\_SAMPLETIME\_1CYCLE5\_SMPR2ALLCHANNELS

ADC\_SAMPLETIME\_7CYCLES5\_SMPR2ALLCHANNELS

ADC\_SAMPLETIME\_13CYCLES5\_SMPR2ALLCHANNELS

ADC\_SAMPLETIME\_28CYCLES5\_SMPR2ALLCHANNELS

ADC\_SAMPLETIME\_41CYCLES5\_SMPR2ALLCHANNELS

ADC\_SAMPLETIME\_55CYCLES5\_SMPR2ALLCHANNELS

ADC\_SAMPLETIME\_71CYCLES5\_SMPR2ALLCHANNELS

ADC\_SAMPLETIME\_239CYCLES5\_SMPR2ALLCHANNELS

ADC\_SAMPLETIME\_1CYCLE5\_SMPR1ALLCHANNELS

ADC\_SAMPLETIME\_7CYCLES5\_SMPR1ALLCHANNELS

ADC\_SAMPLETIME\_13CYCLES5\_SMPR1ALLCHANNELS

ADC\_SAMPLETIME\_28CYCLES5\_SMPR1ALLCHANNELS

ADC\_SAMPLETIME\_41CYCLES5\_SMPR1ALLCHANNELS

ADC\_SAMPLETIME\_55CYCLES5\_SMPR1ALLCHANNELS

ADC\_SAMPLETIME\_71CYCLES5\_SMPR1ALLCHANNELS

ADC\_SAMPLETIME\_239CYCLES5\_SMPR1ALLCHANNELS

**ADC scan mode**



ADC\_SCAN\_DISABLE

ADC\_SCAN\_ENABLE

## 5 HAL ADC Extension Driver

### 5.1 ADCEX Firmware driver registers structures

#### 5.1.1 ADC\_InjectionConfTypeDef

**ADC\_InjectionConfTypeDef** is defined in the `stm32f1xx_hal_adc_ex.h`

##### Data Fields

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

##### Field Documentation

- ***uint32\_t ADC\_InjectionConfTypeDef::InjectedChannel*** Selection of ADC channel to configure This parameter can be a value of [ADC\\_channels](#) Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability. Note: On STM32F1 devices with several ADC: Only ADC1 can access internal measurement channels (VrefInt/TempSensor) Note: On STM32F10xx8 and STM32F10xxB devices: A low-amplitude voltage glitch may be generated (on ADC input 0) on the PA0 pin, when the ADC is converting with injection trigger. It is advised to distribute the analog channels so that Channel 0 is configured as an injected channel. Refer to errata sheet of these devices for more details.
- ***uint32\_t ADC\_InjectionConfTypeDef::InjectedRank*** Rank in the injected group sequencer This parameter must be a value of [ADCEX\\_injected\\_rank](#) Note: In case of need to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions can be adjusted)
- ***uint32\_t ADC\_InjectionConfTypeDef::InjectedSamplingTime*** Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits). This parameter can be a value of [ADC\\_sampling\\_times](#) Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting. Note: In case of usage of internal measurement channels (VrefInt/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS\_vrefint, TS\_temp (values rough order: 5us to 17.1us min).
- ***uint32\_t ADC\_InjectionConfTypeDef::InjectedOffset*** Defines the offset to be subtracted from the raw converted data (for channels set on injected group only). Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively.

- **`uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion`** Specifies the number of ranks that will be converted within the injected group sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min\_Data = 1 and Max\_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **`HAL_ADCEX_InjectedConfigChannel()`** to configure a channel on injected group can impact the configuration of other channels previously set.
- **`uint32_t ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode`** Specifies whether the conversions sequence of injected group is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE. Note: For injected group, number of discontinuous ranks increment is fixed to one-by-one. Caution: this setting impacts the entire injected group. Therefore, call of **`HAL_ADCEX_InjectedConfigChannel()`** to configure a channel on injected group can impact the configuration of other channels previously set.
- **`uint32_t ADC_InjectionConfTypeDef::AutoInjectedConv`** Enables or disables the selected ADC automatic injected group conversion after regular one. This parameter can be set to ENABLE or DISABLE. Note: To use Automatic injected conversion, discontinuous mode must be disabled ('DiscontinuousConvMode' and 'InjectedDiscontinuousConvMode' set to DISABLE). Note: To use Automatic injected conversion, injected group external triggers must be disabled ('ExternalTrigInjecConv' set to ADC\_SOFTWARE\_START). Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **`HAL_ADCEX_InjectedConfigChannel()`** to configure a channel on injected group can impact the configuration of other channels previously set.
- **`uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv`** Selects the external event used to trigger the conversion start of injected group. If set to ADC\_INJECTED\_SOFTWARE\_START, external triggers are disabled. If set to external trigger source, triggering is on event rising edge. This parameter can be a value of [ADCEX\\_External\\_trigger\\_source\\_Injected](#). Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behaviour in case of another parameter update on the fly). Caution: this setting impacts the entire injected group. Therefore, call of **`HAL_ADCEX_InjectedConfigChannel()`** to configure a channel on injected group can impact the configuration of other channels previously set.

### 5.1.2 ADC\_MultiModeTypeDef

**`ADC_MultiModeTypeDef`** is defined in the `stm32f1xx_hal_adc_ex.h`

#### Data Fields

- **`uint32_t Mode`**

#### Field Documentation

- **`uint32_t ADC_MultiModeTypeDef::Mode`** Configures the ADC to operate in independent or multi mode. This parameter can be a value of **`ADCEX_Common_mode`** Note: In dual mode, a change of channel configuration generates a restart that can produce a loss of synchronization. It is recommended to disable dual mode before any configuration change. Note: In case of simultaneous mode used: Exactly the same sampling time should be configured for the 2 channels that will be sampled simultaneously by ADC1 and ADC2. Note: In case of interleaved mode used: To avoid overlap between conversions, maximum sampling time allowed is 7 ADC clock cycles for fast interleaved mode and 14 ADC clock cycles for slow interleaved mode. Note: Some multimode parameters are fixed on STM32F1 and can be configured on other STM32 devices with several ADC (multimode configuration structure can have additional parameters). The equivalences are:
  - Parameter 'DMAAccessMode': On STM32F1, this parameter is fixed to 1 DMA channel (one DMA channel for both ADC, DMA of ADC master). On other STM32 devices with several ADC, this is equivalent to parameter 'ADC\_DMAACCESSMODE\_12\_10\_BITS'.
  - Parameter 'TwoSamplingDelay': On STM32F1, this parameter is fixed to 7 or 14 ADC clock cycles depending on fast or slow interleaved mode selected. On other STM32 devices with several ADC, this is equivalent to parameter 'ADC\_TWOSAMPLINGDELAY\_7CYCLES' (for fast interleaved mode).

## 5.2 ADCEX Firmware driver API description

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

### 5.2.1 IO operation functions

This section provides functions allowing to:

- Start conversion of injected group.
- Stop conversion of injected group.
- Poll for conversion complete on injected group.
- Get result of injected channel conversion.
- Start conversion of injected group and enable interruptions.
- Stop conversion of injected group and disable interruptions.
- Start multimode and enable DMA transfer.
- Stop multimode and disable ADC DMA transfer.
- Get result of multimode conversion.
- Perform the ADC self-calibration for single or differential ending.
- Get calibration factors for single or differential ending.
- Set calibration factors for single or differential ending.
- **`HAL_ADCEX_Calibration_Start()`**
- **`HAL_ADCEX_InjectedStart()`**
- **`HAL_ADCEX_InjectedStop()`**
- **`HAL_ADCEX_InjectedPollForConversion()`**
- **`HAL_ADCEX_InjectedStart_IT()`**
- **`HAL_ADCEX_InjectedStop_IT()`**
- **`HAL_ADCEX_MultiModeStart_DMA()`**
- **`HAL_ADCEX_MultiModeStop_DMA()`**
- **`HAL_ADCEX_InjectedGetValue()`**
- **`HAL_ADCEX_MultiModeGetValue()`**
- **`HAL_ADCEX_InjectedConvCpltCallback()`**

## 5.2.2 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on injected group
- Configure multimode
- [HAL\\_ADCEX\\_InjectedConfigChannel\(\)](#)
- [HAL\\_ADCEX\\_MultiModeConfigChannel\(\)](#)

## 5.2.3 HAL\_ADCEX\_Calibration\_Start

Function Name	<b>HAL_StatusTypeDef HAL_ADCEX_Calibration_Start (ADC_HandleTypeDef * hadc)</b>
Function Description	Perform an ADC automatic self-calibration Calibration prerequisite: ADC must be disabled (execute this function before HAL_ADC_Start() or after HAL_ADC_Stop() ).
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 5.2.4 HAL\_ADCEX\_InjectedStart

Function Name	<b>HAL_StatusTypeDef HAL_ADCEX_InjectedStart (ADC_HandleTypeDef * hadc)</b>
Function Description	Enables ADC, starts conversion of injected group.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 5.2.5 HAL\_ADCEX\_InjectedStop

Function Name	<b>HAL_StatusTypeDef HAL_ADCEX_InjectedStop (ADC_HandleTypeDef * hadc)</b>
Function Description	Stop conversion of injected channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.</li> <li>• In case of auto-injection mode, HAL_ADC_Stop must be used.</li> </ul>

## 5.2.6 HAL\_ADCEX\_InjectedPollForConversion

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion</b> (ADC_HandleTypeDef * hadc, uint32_t Timeout)
Function Description	Wait for injected group conversion to be completed.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b>: ADC handle</li><li>• <b>Timeout</b>: Timeout value in millisecond.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 5.2.7 HAL\_ADCEx\_InjectedStart\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT</b> (ADC_HandleTypeDef * hadc)
Function Description	Enables ADC, starts conversion of injected group with interruption.

### 5.2.8 HAL\_ADCEx\_InjectedStop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT</b> (ADC_HandleTypeDef * hadc)
Function Description	Stop conversion of injected channels, disable interruption of end-of-conversion.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b>: ADC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>
Notes	<ul style="list-style-type: none"><li>• If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.</li></ul>

### 5.2.9 HAL\_ADCEx\_MultiModeStart\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA</b> (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)
Function Description	Enables ADC, starts conversion of regular group and transfers result through DMA.

### 5.2.10 HAL\_ADCEx\_MultiModeStop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA</b> (ADC_HandleTypeDef * hadc)
Function Description	Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b>: ADC handle of ADC master (handle of ADC slave must not be used)</li></ul>

Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Multimode is kept enabled after this function. To disable multimode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_ReInit().</li> <li>• In case of DMA configured in circular mode, function HAL_ADC_Stop_DMA must be called after this function with handle of ADC slave, to properly disable the DMA channel.</li> </ul>

### 5.2.11 HAL\_ADCEx\_InjectedGetValue

Function Name	<b>uint32_t HAL_ADCEx_InjectedGetValue (ADC_HandleTypeDef * hadc, uint32_t InjectedRank)</b>
Function Description	Get ADC injected group conversion result.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> <li>• <b>InjectedRank</b>: the converted ADC injected rank. This parameter can be one of the following values:  ADC_INJECTED_RANK_1: Injected Channel1 selected  ADC_INJECTED_RANK_2: Injected Channel2 selected  ADC_INJECTED_RANK_3: Injected Channel3 selected  ADC_INJECTED_RANK_4: Injected Channel4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 5.2.12 HAL\_ADCEx\_MultiModeGetValue

Function Name	<b>uint32_t HAL_ADCEx_MultiModeGetValue (ADC_HandleTypeDef * hadc)</b>
Function Description	Returns the last ADC Master&Slave regular conversions results data in the selected multi mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle of ADC master (handle of ADC slave must not be used)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• The converted data value.</li> </ul>

### 5.2.13 HAL\_ADCEx\_InjectedConvCpltCallback

Function Name	<b>void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)</b>
Function Description	Injected conversion complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 5.2.14 HAL\_ADCEx\_InjectedConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel</b> (ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef * sConfigInjected)
Function Description	Configures the ADC injected group and the selected channel to be linked to the injected group.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> <li>• <b>sConfigInjected</b>: Structure of ADC injected group and ADC channel for injected group.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Possibility to update parameters on the fly: This function initializes injected group, following calls to this function can be used to reconfigure some parameters of structure "ADC_InjectionConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: this function must be called when ADC is not under conversion.</li> </ul>

### 5.2.15 HAL\_ADCEx\_MultiModeConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel</b> (ADC_HandleTypeDef * hadc, ADC_MultiModeTypeDef * multimode)
Function Description	Enable ADC multimode and configure multimode parameters.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b>: ADC handle</li> <li>• <b>multimode</b>: Structure of ADC multimode configuration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Possibility to update parameters on the fly: This function initializes multimode parameters, following calls to this function can be used to reconfigure some parameters of structure "ADC_MultiModeTypeDef" on the fly, without resetting the ADCs (both ADCs of the common group). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_MultiModeTypeDef".</li> <li>• To change back configuration from multimode to single mode, ADC must be reset (using function HAL_ADC_Init() ).</li> </ul>

## 5.3 ADCEx Firmware driver defines

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

### 5.3.1 ADCEx

ADCEx

**ADC Extended Dual ADC Mode**

ADC\_MODE\_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

ADC\_DUALMODE\_REGSIMULT\_INJECSIMU

ADC dual mode enabled: Combined



LT	regular simultaneous + injected simultaneous mode
ADC_DUALMODE_REGSIMULT_ALTERTRIG	ADC dual mode enabled: Combined regular simultaneous + alternate trigger mode
ADC_DUALMODE_INJECSIMULT_INTERLFAST	ADC dual mode enabled: Combined injected simultaneous + fast interleaved mode (delay between ADC sampling phases: 7 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_INJECSIMULT_INTERLSLOW	ADC dual mode enabled: Combined injected simultaneous + slow Interleaved mode (delay between ADC sampling phases: 14 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_INJECSIMULT	ADC dual mode enabled: Injected simultaneous mode only
ADC_DUALMODE_REGSIMULT	ADC dual mode enabled: Regular simultaneous mode only
ADC_DUALMODE_INTERLFAST	ADC dual mode enabled: Fast interleaved mode only (delay between ADC sampling phases: 7 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_INTERLSLOW	ADC dual mode enabled: Slow interleaved mode only (delay between ADC sampling phases: 14 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_ALTERTRIG	ADC dual mode enabled: Alternate trigger mode only
<b>ADCEx external trigger enable for injected group</b>	
ADC_EXTERNALTRIGINJECCONV_EDGE_NONE	
ADC_EXTERNALTRIGINJECCONV_EDGE_RISING	
<b>ADCEx External trigger selection for injected group</b>	
ADC_EXTERNALTRIGINJECCONV_T2_TRGO	< List of external triggers with generic trigger name, independently of
ADC_EXTERNALTRIGINJECCONV_T2_CC1	
ADC_EXTERNALTRIGINJECCONV_T3_CC4	

ADC\_EXTERNALTRIGINJEC\_CONV\_T4\_TRGO  
ADC\_EXTERNALTRIGINJEC\_CONV\_EXT\_IT15      External triggers of injected group for ADC3 only  
  
ADC\_EXTERNALTRIGINJEC\_CONV\_T4\_CC3  
ADC\_EXTERNALTRIGINJEC\_CONV\_T8\_CC2  
ADC\_EXTERNALTRIGINJEC\_CONV\_T5\_TRGO  
ADC\_EXTERNALTRIGINJEC\_CONV\_T5\_CC4  
ADC\_EXTERNALTRIGINJEC\_CONV\_T1\_CC4      < External triggers of injected group for all ADC instances  
  
ADC\_EXTERNALTRIGINJEC\_CONV\_T1\_TRGO      Note: TIM8\_CC4 is available on ADC1 and ADC2 only in high-density and  
  
ADC\_EXTERNALTRIGINJEC\_CONV\_T8\_CC4  
ADC\_INJECTED\_SOFTWARE\_START  
**ADCEx injected nb conv verification**  
IS\_ADC\_INJECTED\_NB\_CONV  
**ADCEx rank into injected group**  
ADC\_INJECTED\_RANK\_1  
ADC\_INJECTED\_RANK\_2  
ADC\_INJECTED\_RANK\_3  
ADC\_INJECTED\_RANK\_4  
**ADC Extended Internal HAL driver trigger selection for injected group**  
ADC1\_2\_EXTERNALTRIGINJEC\_T2\_TRGO  
ADC1\_2\_EXTERNALTRIGINJEC\_T2\_CC1  
ADC1\_2\_EXTERNALTRIGINJEC\_T3\_CC4  
ADC1\_2\_EXTERNALTRIGINJEC\_T4\_TRGO  
ADC1\_2\_EXTERNALTRIGINJEC\_EXT\_IT15  
ADC1\_2\_EXTERNALTRIGINJEC\_T8\_CC4  
ADC3\_EXTERNALTRIGINJEC\_T4\_CC3  
ADC3\_EXTERNALTRIGINJEC\_T8\_CC2  
ADC3\_EXTERNALTRIGINJEC\_T8\_CC4  
ADC3\_EXTERNALTRIGINJEC\_T5\_TRGO  
ADC3\_EXTERNALTRIGINJEC\_T5\_CC4  
ADC1\_2\_3\_EXTERNALTRIGINJEC\_T1\_TRGO  
ADC1\_2\_3\_EXTERNALTRIGINJEC\_T1\_CC4  
ADC1\_2\_3\_JSWSTART  
**ADC Extended Internal HAL driver trigger selection for regular group**  
ADC1\_2\_EXTERNALTRIG\_T1\_CC1

ADC1\_2\_EXTERNALTRIG\_T1\_CC2  
 ADC1\_2\_EXTERNALTRIG\_T2\_CC2  
 ADC1\_2\_EXTERNALTRIG\_T3\_TRGO  
 ADC1\_2\_EXTERNALTRIG\_T4\_CC4  
 ADC1\_2\_EXTERNALTRIG\_EXT\_IT11  
 ADC1\_2\_EXTERNALTRIG\_T8\_TRGO  
 ADC3\_EXTERNALTRIG\_T3\_CC1  
 ADC3\_EXTERNALTRIG\_T2\_CC3  
 ADC3\_EXTERNALTRIG\_T8\_CC1  
 ADC3\_EXTERNALTRIG\_T8\_TRGO  
 ADC3\_EXTERNALTRIG\_T5\_CC1  
 ADC3\_EXTERNALTRIG\_T5\_CC3  
 ADC1\_2\_3\_EXTERNALTRIG\_T1\_CC3  
 ADC1\_2\_3\_SWSTART

#### **ADCEx Private Constants**

ADC\_PRECALIBRATION\_DELAY\_ADCCLOCKCYCLES  
 ADC\_CALIBRATION\_TIMEOUT  
 ADC\_TEMPSENSOR\_DELAY\_US

#### **ADCEx Private Macro**

ADC\_CFGR\_EXTSEL

#### **Description:**

- For devices with 3 ADCs:  
Defines the external trigger source for regular group according to ADC into common group ADC1&ADC2 or ADC3 (some triggers with same source have different value to be programmed into ADC EXTSEL bits of CR2 register).

#### **Parameters:**

- `__HANDLE__`: ADC handle
- `__EXT_TRIG_CONV__`:  
External trigger selected for regular group.

#### **Return value:**

- External: trigger to be programmed into EXTSEL bits of CR2 register

#### **Description:**

- For devices with 3 ADCs:  
Defines the external trigger source for injected group

ADC\_CFGR\_JEXTSEL

according to ADC into common group ADC1&ADC2 or ADC3 (some triggers with same source have different value to be programmed into ADC JEXTSEL bits of CR2 register).

**Parameters:**

- `__HANDLE__`: ADC handle
- `__EXT_TRIG_INJECTCONV__`: External trigger selected for injected group.

**Return value:**

- External: trigger to be programmed into JEXTSEL bits of CR2 register

**Description:**

- Verification if multimode is enabled for the selected ADC (multimode ADC master or ADC slave) (applicable for devices with several ADCs)

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- Multimode: state: RESET if multimode is disabled, other value if multimode is enabled

**Description:**

- Verification of condition for ADC start conversion: ADC must be in non-multimode, or multimode with handle of ADC master (applicable for devices with several ADCs)

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None:

**Description:**

- Set handle of the other ADC sharing the common multimode settings.

**Parameters:**

- `__HANDLE__`: ADC handle

`ADC_MULTIMODE_IS_ENABLE`

`ADC_NONMULTIMODE_OR_MULTIMODEMASTER`

`ADC_COMMON_ADC_OTHER`

ADC\_MULTI\_SLAVE

- `__HANDLE_OTHER_ADC__`: other ADC handle

**Return value:**

- None:

**Description:**

- Set handle of the ADC slave associated to the ADC master  
On STM32F1 devices, ADC slave is always ADC2 (this can be different on other STM32 devices)

**Parameters:**

- `__HANDLE_MASTER__`: ADC master handle
- `__HANDLE_SLAVE__`: ADC slave handle

**Return value:**

- None:

IS\_ADC\_INJECTED\_RANK  
IS\_ADC\_EXTTRIGINJEC\_EDGE  
IS\_ADC\_EXTTRIG  
IS\_ADC\_EXTTRIGINJEC  
IS\_ADC\_MODE

## 6 HAL CAN Generic Driver

### 6.1 CAN Firmware driver registers structures

#### 6.1.1 CAN\_InitTypeDef

**CAN\_InitTypeDef** is defined in the stm32f1xx\_hal\_can.h

##### Data Fields

- **uint32\_t Prescaler**
- **uint32\_t Mode**
- **uint32\_t SJW**
- **uint32\_t BS1**
- **uint32\_t BS2**
- **uint32\_t TTCM**
- **uint32\_t ABOM**
- **uint32\_t AWUM**
- **uint32\_t NART**
- **uint32\_t RFLM**
- **uint32\_t TXFP**

##### Field Documentation

- **uint32\_t CAN\_InitTypeDef::Prescaler** Specifies the length of a time quantum. This parameter must be a number between Min\_Data = 1 and Max\_Data = 1024.
- **uint32\_t CAN\_InitTypeDef::Mode** Specifies the CAN operating mode. This parameter can be a value of [CAN\\_operating\\_mode](#)
- **uint32\_t CAN\_InitTypeDef::SJW** Specifies the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter can be a value of [CAN\\_synchronisation\\_jump\\_width](#)
- **uint32\_t CAN\_InitTypeDef::BS1** Specifies the number of time quanta in Bit Segment 1. This parameter can be a value of [CAN\\_time\\_quantum\\_in\\_bit\\_segment\\_1](#)
- **uint32\_t CAN\_InitTypeDef::BS2** Specifies the number of time quanta in Bit Segment 2. This parameter can be a value of [CAN\\_time\\_quantum\\_in\\_bit\\_segment\\_2](#)
- **uint32\_t CAN\_InitTypeDef::TTCM** Enable or disable the time triggered communication mode. This parameter can be set to ENABLE or DISABLE.
- **uint32\_t CAN\_InitTypeDef::ABOM** Enable or disable the automatic bus-off management. This parameter can be set to ENABLE or DISABLE.
- **uint32\_t CAN\_InitTypeDef::AWUM** Enable or disable the automatic wake-up mode. This parameter can be set to ENABLE or DISABLE.
- **uint32\_t CAN\_InitTypeDef::NART** Enable or disable the non-automatic retransmission mode. This parameter can be set to ENABLE or DISABLE.
- **uint32\_t CAN\_InitTypeDef::RFLM** Enable or disable the Receive FIFO Locked mode. This parameter can be set to ENABLE or DISABLE.
- **uint32\_t CAN\_InitTypeDef::TXFP** Enable or disable the transmit FIFO priority. This parameter can be set to ENABLE or DISABLE.

#### 6.1.2 CanTxMsgTypeDef

**CanTxMsgTypeDef** is defined in the stm32f1xx\_hal\_can.h

**Data Fields**

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

**Field Documentation**

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

**6.1.3 CanRxMsgTypeDef**

*CanRxMsgTypeDef* is defined in the stm32f1xx\_hal\_can.h

**Data Fields**

- *uint32\_t StdId*
- *uint32\_t ExtId*
- *uint32\_t IDE*
- *uint32\_t RTR*
- *uint32\_t DLC*
- *uint32\_t Data*
- *uint32\_t FMI*
- *uint32\_t FIFONumber*

**Field Documentation**

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

- **`uint32_t CanRxMsgTypeDef::DLC`** Specifies the length of the frame that will be received. This parameter must be a number between `Min_Data = 0` and `Max_Data = 8`.
- **`uint32_t CanRxMsgTypeDef::Data[8]`** Contains the data to be received. This parameter must be a number between `Min_Data = 0` and `Max_Data = 0xFF`.
- **`uint32_t CanRxMsgTypeDef::FMI`** Specifies the index of the filter the message stored in the mailbox passes through. This parameter must be a number between `Min_Data = 0` and `Max_Data = 0xFF`.
- **`uint32_t CanRxMsgTypeDef::FIFONumber`** Specifies the receive FIFO number. This parameter can be a value of [CAN\\_receive\\_FIFO\\_number\\_constants](#)

#### 6.1.4 CAN\_HandleTypeDef

**`CAN_HandleTypeDef`** is defined in the `stm32f1xx_hal_can.h`

##### Data Fields

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

##### Field Documentation

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

## 6.2 CAN Firmware driver API description

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

### 6.2.1 How to use this driver

1. Enable the CAN controller interface clock using `__HAL_RCC_CAN1_CLK_ENABLE()` for CAN1 and `__HAL_RCC_CAN2_CLK_ENABLE()` for CAN2. In case you are using CAN2 only, you have to enable the CAN1 clock.
2. CAN pins configuration
  - Enable the clock for the CAN GPIOs using the following function:  
`__HAL_RCC_GPIOx_CLK_ENABLE();`
  - Connect and configure the involved CAN pins using the following function  
`HAL_GPIO_Init();`



3. Initialise and configure the CAN using HAL\_CAN\_Init() function.
4. Transmit the desired CAN frame using HAL\_CAN\_Transmit() function.
5. Receive a CAN frame using HAL\_CAN\_Receive() function.

### Polling mode IO operation

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

### Interrupt mode IO operation

- Start the CAN peripheral transmission using HAL\_CAN\_Transmit\_IT()
- Start the CAN peripheral reception using HAL\_CAN\_Receive\_IT()
- Use HAL\_CAN\_IRQHandler() called under the used CAN Interrupt subroutine
- At CAN end of transmission HAL\_CAN\_TxCpltCallback() function is executed and user can add his own code by customization of function pointer HAL\_CAN\_TxCpltCallback
- In case of CAN Error, HAL\_CAN\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_CAN\_ErrorCallback

### CAN HAL driver macros list

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

- \_\_HAL\_CAN\_ENABLE\_IT: Enable the specified CAN interrupts
- \_\_HAL\_CAN\_DISABLE\_IT: Disable the specified CAN interrupts
- \_\_HAL\_CAN\_GET\_IT\_SOURCE: Check if the specified CAN interrupt source is enabled or disabled
- \_\_HAL\_CAN\_CLEAR\_FLAG: Clear the CAN's pending flags
- \_\_HAL\_CAN\_GET\_FLAG: Get the selected CAN's flag status



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

## 6.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the CAN.
- De-initialize the CAN.
- [\*\*HAL\\_CAN\\_Init\(\)\*\*](#)
- [\*\*HAL\\_CAN\\_ConfigFilter\(\)\*\*](#)
- [\*\*HAL\\_CAN\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_CAN\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_CAN\\_MspDeInit\(\)\*\*](#)

## 6.2.3 IO operation functions

This section provides functions allowing to:

- Transmit a CAN frame message.
- Receive a CAN frame message.
- Enter CAN peripheral in sleep mode.
- Wake up the CAN peripheral from sleep mode.
- [HAL\\_CAN\\_Transmit\(\)](#)
- [HAL\\_CAN\\_Transmit\\_IT\(\)](#)
- [HAL\\_CAN\\_Receive\(\)](#)
- [HAL\\_CAN\\_Receive\\_IT\(\)](#)
- [HAL\\_CAN\\_Sleep\(\)](#)
- [HAL\\_CAN\\_WakeUp\(\)](#)
- [HAL\\_CAN\\_IRQHandler\(\)](#)
- [HAL\\_CAN\\_TxCpltCallback\(\)](#)
- [HAL\\_CAN\\_RxCpltCallback\(\)](#)
- [HAL\\_CAN\\_ErrorCallback\(\)](#)

## 6.2.4 Peripheral State and Error functions

This subsection provides functions allowing to :

- Check the CAN state.
- Check CAN Errors detected during interrupt process
- [HAL\\_CAN\\_GetState\(\)](#)
- [HAL\\_CAN\\_GetError\(\)](#)

## 6.2.5 HAL\_CAN\_Init

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Init (CAN_HandleTypeDef * hcan)</b>
Function Description	Initializes the CAN peripheral according to the specified parameters in the CAN_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcan</b>: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 6.2.6 HAL\_CAN\_ConfigFilter

Function Name	<b>HAL_StatusTypeDef HAL_CAN_ConfigFilter (CAN_HandleTypeDef * hcan, CAN_FilterConfTypeDef * sFilterConfig)</b>
Function Description	Configures the CAN reception filter according to the specified parameters in the CAN_FilterInitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcan</b>: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> <li>• <b>sFilterConfig</b>: pointer to a CAN_FilterConfTypeDef structure</li> </ul>

that contains the filter configuration information.

Return values

- None

## 6.2.7 HAL\_CAN\_DeInit

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

Function Description Deinitializes the CANx peripheral registers to their default reset values.

Parameters

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

Return values

- HAL status

## 6.2.8 HAL\_CAN\_MspInit

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

Function Description Initializes the CAN MSP.

Parameters

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

Return values

- None

## 6.2.9 HAL\_CAN\_MspDeInit

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

Function Description DeInitializes the CAN MSP.

Parameters

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

Return values

- None

## 6.2.10 HAL\_CAN\_Transmit

Function Name **HAL\_StatusTypeDef HAL\_CAN\_Transmit (CAN\_HandleTypeDef \* hcan, uint32\_t Timeout)**

Function Description Initiates and transmits a CAN frame message.

Parameters

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

Return values

- HAL status

## 6.2.11 HAL\_CAN\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Transmit_IT (CAN_HandleTypeDef * hcan)</b>
Function Description	Initiates and transmits a CAN frame message.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b>: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 6.2.12 HAL\_CAN\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Receive (CAN_HandleTypeDef * hcan, uint8_t FIFONumber, uint32_t Timeout)</b>
Function Description	Receives a correct CAN frame.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b>: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li><li>• <b>FIFONumber</b>: FIFO Number value</li><li>• <b>Timeout</b>: Specify Timeout value</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li><li>• None</li></ul>

### 6.2.13 HAL\_CAN\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Receive_IT (CAN_HandleTypeDef * hcan, uint8_t FIFONumber)</b>
Function Description	Receives a correct CAN frame.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b>: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li><li>• <b>FIFONumber</b>: Specify the FIFO number</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li><li>• None</li></ul>

### 6.2.14 HAL\_CAN\_Sleep

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Sleep (CAN_HandleTypeDef * hcan)</b>
Function Description	Enters the Sleep (low power) mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b>: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status.</li></ul>

### 6.2.15 HAL\_CAN\_WakeUp

Function Name	<b>HAL_StatusTypeDef HAL_CAN_WakeUp (CAN_HandleTypeDef * hcan)</b>
Function Description	Wakes up the CAN peripheral from sleep mode, after that the CAN peripheral is in the normal mode.
Parameters	<ul style="list-style-type: none"> <li><b>hcan:</b> pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status.</li> </ul>

### 6.2.16 HAL\_CAN\_IRQHandler

Function Name	<b>void HAL_CAN_IRQHandler (CAN_HandleTypeDef * hcan)</b>
Function Description	Handles CAN interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hcan:</b> pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 6.2.17 HAL\_CAN\_TxCpltCallback

Function Name	<b>void HAL_CAN_TxCpltCallback (CAN_HandleTypeDef * hcan)</b>
Function Description	Transmission complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hcan:</b> pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 6.2.18 HAL\_CAN\_RxCpltCallback

Function Name	<b>void HAL_CAN_RxCpltCallback (CAN_HandleTypeDef * hcan)</b>
Function Description	Transmission complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hcan:</b> pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 6.2.19 HAL\_CAN\_ErrorCallback

Function Name	<b>void HAL_CAN_ErrorCallback (CAN_HandleTypeDef * hcan)</b>
Function Description	Error CAN callback.
Parameters	<ul style="list-style-type: none"> <li><b>hcan:</b> pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**6.2.20 HAL\_CAN\_GetState**

Function Name	<b>HAL_CAN_StateTypeDef HAL_CAN_GetState (CAN_HandleTypeDef * hcan)</b>
Function Description	return the CAN state
Parameters	<ul style="list-style-type: none"> <li><b>hcan:</b> pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

**6.2.21 HAL\_CAN\_GetError**

Function Name	<b>uint32_t HAL_CAN_GetError (CAN_HandleTypeDef * hcan)</b>
Function Description	Return the CAN error code.
Parameters	<ul style="list-style-type: none"> <li><b>hcan:</b> pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>CAN Error Code</li> </ul>

**6.3 CAN Firmware driver defines**

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

**6.3.1 CAN**

CAN

**CAN Error Code**

HAL_CAN_ERROR_NONE	No error
HAL_CAN_ERROR_EWG	EWG error
HAL_CAN_ERROR_EPV	EPV error
HAL_CAN_ERROR_BOF	BOF error
HAL_CAN_ERROR_STF	Stuff error
HAL_CAN_ERROR_FOR	Form error
HAL_CAN_ERROR_ACK	Acknowledgment error
HAL_CAN_ERROR_BR	Bit recessive
HAL_CAN_ERROR_BD	LEC dominant
HAL_CAN_ERROR_CRC	LEC transfer error

**CAN Exported Macros**

<b>__HAL_CAN_RESET_HANDLE_STATE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset CAN handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__:</b> CAN handle.</li> </ul>
-------------------------------------	---

**\_\_HAL\_CAN\_ENABLE\_IT****Return value:**

- None:

**Description:**

- Enable the specified CAN interrupts.

**Parameters:**

- \_\_HANDLE\_\_: CAN handle.
- \_\_INTERRUPT\_\_: CAN Interrupt. This parameter can be one of the following values:
  - CAN\_IT\_TME: Transmit mailbox empty interrupt enable
  - CAN\_IT\_FMP0: FIFO 0 message pending interrupt
  - CAN\_IT\_FF0 : FIFO 0 full interrupt
  - CAN\_IT\_FOV0: FIFO 0 overrun interrupt
  - CAN\_IT\_FMP1: FIFO 1 message pending interrupt
  - CAN\_IT\_FF1 : FIFO 1 full interrupt
  - CAN\_IT\_FOV1: FIFO 1 overrun interrupt
  - CAN\_IT\_WKU : Wake-up interrupt
  - CAN\_IT\_SLK : Sleep acknowledge interrupt
  - CAN\_IT\_EWG : Error warning interrupt
  - CAN\_IT\_EPV : Error passive interrupt
  - CAN\_IT\_BOF : Bus-off interrupt
  - CAN\_IT\_LEC : Last error code interrupt
  - CAN\_IT\_ERR : Error Interrupt

**Return value:**

- None.:

**Description:**

- Disable the specified CAN interrupts.

**Parameters:**

- \_\_HANDLE\_\_: CAN handle.
- \_\_INTERRUPT\_\_: CAN Interrupt. This parameter can be one of the following values:
  - CAN\_IT\_TME: Transmit mailbox empty interrupt enable
  - CAN\_IT\_FMP0: FIFO 0 message pending interrupt
  - CAN\_IT\_FF0 : FIFO 0 full interrupt
  - CAN\_IT\_FOV0: FIFO 0 overrun interrupt
  - CAN\_IT\_FMP1: FIFO 1 message

**\_\_HAL\_CAN\_DISABLE\_IT**

- pending interrupt
- CAN\_IT\_FF1 : FIFO 1 full interrupt
- CAN\_IT\_FOV1: FIFO 1 overrun interrupt
- CAN\_IT\_WKU : Wake-up interrupt
- CAN\_IT\_SLK : Sleep acknowledge interrupt
- CAN\_IT\_EWG : Error warning interrupt
- CAN\_IT\_EPV : Error passive interrupt
- CAN\_IT\_BOF : Bus-off interrupt
- CAN\_IT\_LEC : Last error code interrupt
- CAN\_IT\_ERR : Error Interrupt

**Return value:**

- None.:

**\_\_HAL\_CAN\_MSG\_PENDING****Description:**

- Return the number of pending received messages.

**Parameters:**

- \_\_HANDLE\_\_: CAN handle.
- \_\_FIFONUMBER\_\_: Receive FIFO number, CAN\_FIFO0 or CAN\_FIFO1.

**Return value:**

- The: number of pending message.

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

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the CAN Handle.
- \_\_FLAG\_\_: specifies the flag to check.  
This parameter can be one of the following values:
  - CAN\_TSR\_RQCP0: Request MailBox0 Flag
  - CAN\_TSR\_RQCP1: Request MailBox1 Flag
  - CAN\_TSR\_RQCP2: Request MailBox2 Flag
  - CAN\_FLAG\_TXOK0: Transmission OK MailBox0 Flag
  - CAN\_FLAG\_TXOK1: Transmission OK MailBox1 Flag
  - CAN\_FLAG\_TXOK2: Transmission OK MailBox2 Flag
  - CAN\_FLAG\_TME0: Transmit mailbox 0 empty Flag
  - CAN\_FLAG\_TME1: Transmit mailbox



- 1 empty Flag
- CAN\_FLAG\_TME2: Transmit mailbox 2 empty Flag
- CAN\_FLAG\_FMP0: FIFO 0 Message Pending Flag
- CAN\_FLAG\_FF0: FIFO 0 Full Flag
- CAN\_FLAG\_FOV0: FIFO 0 Overrun Flag
- CAN\_FLAG\_FMP1: FIFO 1 Message Pending Flag
- CAN\_FLAG\_FF1: FIFO 1 Full Flag
- CAN\_FLAG\_FOV1: FIFO 1 Overrun Flag
- CAN\_FLAG\_WKU: Wake up Flag
- CAN\_FLAG\_SLAK: Sleep acknowledge Flag
- CAN\_FLAG\_SLAKI: Sleep acknowledge Flag
- CAN\_FLAG\_EWG: Error Warning Flag
- CAN\_FLAG\_EPV: Error Passive Flag
- CAN\_FLAG\_BOF: Bus-Off Flag

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

**Description:**

- Clear the specified CAN pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the CAN Handle.
- \_\_FLAG\_\_: specifies the flag to check.  
This parameter can be one of the following values:
  - CAN\_TSR\_RQCP0: Request MailBox0 Flag
  - CAN\_TSR\_RQCP1: Request MailBox1 Flag
  - CAN\_TSR\_RQCP2: Request MailBox2 Flag
  - CAN\_FLAG\_TXOK0: Transmission OK MailBox0 Flag
  - CAN\_FLAG\_TXOK1: Transmission OK MailBox1 Flag
  - CAN\_FLAG\_TXOK2: Transmission OK MailBox2 Flag
  - CAN\_FLAG\_TME0: Transmit mailbox 0 empty Flag
  - CAN\_FLAG\_TME1: Transmit mailbox 1 empty Flag
  - CAN\_FLAG\_TME2: Transmit mailbox 2 empty Flag
  - CAN\_FLAG\_FMP0: FIFO 0 Message

\_\_HAL\_CAN\_CLEAR\_FLAG

- Pending Flag
- CAN\_FLAG\_FF0: FIFO 0 Full Flag
- CAN\_FLAG\_FOV0: FIFO 0 Overrun Flag
- CAN\_FLAG\_FMP1: FIFO 1 Message Pending Flag
- CAN\_FLAG\_FF1: FIFO 1 Full Flag
- CAN\_FLAG\_FOV1: FIFO 1 Overrun Flag
- CAN\_FLAG\_WKU: Wake up Flag
- CAN\_FLAG\_SLAKI: Sleep acknowledge Flag

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

**\_\_HAL\_CAN\_GET\_IT\_SOURCE****Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the CAN Handle.
- \_\_INTERRUPT\_\_: specifies the CAN interrupt source to check. This parameter can be one of the following values:
  - CAN\_IT\_TME: Transmit mailbox empty interrupt enable
  - CAN\_IT\_FMP0: FIFO 0 message pending interrupt
  - CAN\_IT\_FF0 : FIFO 0 full interrupt
  - CAN\_IT\_FOV0: FIFO 0 overrun interrupt
  - CAN\_IT\_FMP1: FIFO 1 message pending interrupt
  - CAN\_IT\_FF1 : FIFO 1 full interrupt
  - CAN\_IT\_FOV1: FIFO 1 overrun interrupt
  - CAN\_IT\_WKU : Wake-up interrupt
  - CAN\_IT\_SLK : Sleep acknowledge interrupt
  - CAN\_IT\_EWG : Error warning interrupt
  - CAN\_IT\_EPV : Error passive interrupt
  - CAN\_IT\_BOF : Bus-off interrupt
  - CAN\_IT\_LEC : Last error code interrupt
  - CAN\_IT\_ERR : Error Interrupt

**Return value:**

- The: new state of \_\_IT\_\_ (TRUE or FALSE).

**\_\_HAL\_CAN\_TRANSMIT\_STATUS****Description:**

- Check the transmission status of a CAN Frame.

**Parameters:**

- `__HANDLE__`: specifies the CAN Handle.
- `__TRANSMITMAILBOX__`: the number of the mailbox that is used for transmission.

**Return value:**

- The: new status of transmission (TRUE or FALSE).

**Description:**

- Release the specified receive FIFO.

**Parameters:**

- `__HANDLE__`: CAN handle.
- `__FIFONUMBER__`: Receive FIFO number, CAN\_FIFO0 or CAN\_FIFO1.

**Return value:**

- None.:

**Description:**

- Cancel a transmit request.

**Parameters:**

- `__HANDLE__`: specifies the CAN Handle.
- `__TRANSMITMAILBOX__`: the number of the mailbox that is used for transmission.

**Return value:**

- None.:

**Description:**

- Enable or disables the DBG Freeze for CAN.

**Parameters:**

- `__HANDLE__`: specifies the CAN Handle.
- `__NEWSTATE__`: new state of the CAN peripheral. This parameter can be: ENABLE (CAN reception/transmission is frozen during debug. Reception FIFOs can still be accessed/controlled normally) or DISABLE (CAN is working during debug).

**Return value:**

- None:

`__HAL_CAN_FIFO_RELEASE``__HAL_CAN_CANCEL_TRANSMIT``__HAL_CAN_DBG_FREEZE`**CAN Filter FIFO**

CAN\_FILTER\_FIFO0 Filter FIFO 0 assignment for filter x

CAN\_FILTER\_FIFO1 Filter FIFO 1 assignment for filter x

**CAN Filter Mode**

CAN\_FILTERMODE\_IDMASK Identifier mask mode

CAN\_FILTERMODE\_IDLIST Identifier list mode

**CAN Filter Scale**

CAN\_FILTERSCALE\_16BIT Two 16-bit filters

CAN\_FILTERSCALE\_32BIT One 32-bit filter

**CAN Flags**

CAN\_FLAG\_RQCP0 Request MailBox0 flag

CAN\_FLAG\_RQCP1 Request MailBox1 flag

CAN\_FLAG\_RQCP2 Request MailBox2 flag

CAN\_FLAG\_TXOK0 Transmission OK MailBox0 flag

CAN\_FLAG\_TXOK1 Transmission OK MailBox1 flag

CAN\_FLAG\_TXOK2 Transmission OK MailBox2 flag

CAN\_FLAG\_TME0 Transmit mailbox 0 empty flag

CAN\_FLAG\_TME1 Transmit mailbox 0 empty flag

CAN\_FLAG\_TME2 Transmit mailbox 0 empty flag

CAN\_FLAG\_FF0 FIFO 0 Full flag

CAN\_FLAG\_FOV0 FIFO 0 Overrun flag

CAN\_FLAG\_FF1 FIFO 1 Full flag

CAN\_FLAG\_FOV1 FIFO 1 Overrun flag

CAN\_FLAG\_WKU Wake up flag

CAN\_FLAG\_SLAKE Sleep acknowledge flag

CAN\_FLAG\_SLAKEI Sleep acknowledge flag

CAN\_FLAG\_EWG Error warning flag

CAN\_FLAG\_EPV Error passive flag

CAN\_FLAG\_BOFF Bus-Off flag

**CAN Identifier Type**

CAN\_ID\_STD Standard Id

CAN\_ID\_EXT Extended Id

**CAN initialization Status**

CAN\_INITSTATUS\_FAILED CAN initialization failed

CAN\_INITSTATUS\_SUCCESS CAN initialization OK

**CAN Interrupts**

CAN\_IT\_TME Transmit mailbox empty interrupt

CAN\_IT\_FMP0 FIFO 0 message pending interrupt

CAN\_IT\_FF0 FIFO 0 full interrupt

---

CAN_IT_FOV0	FIFO 0 overrun interrupt
CAN_IT_FMP1	FIFO 1 message pending interrupt
CAN_IT_FF1	FIFO 1 full interrupt
CAN_IT_FOV1	FIFO 1 overrun interrupt
CAN_IT_WKU	Wake-up interrupt
CAN_IT_SLK	Sleep acknowledge interrupt
CAN_IT_EWG	Error warning interrupt
CAN_IT_EPV	Error passive interrupt
CAN_IT_BOF	Bus-off interrupt
CAN_IT_LEC	Last error code interrupt
CAN_IT_ERR	Error Interrupt

**CAN Operating Mode**

CAN_MODE_NORMAL	Normal mode
CAN_MODE_LOOPBACK	Loopback mode
CAN_MODE_SILENT	Silent mode
CAN_MODE_SILENT_LOOPBACK	Loopback combined with silent mode

**CAN Private Constants**

CAN\_TIMEOUT\_VALUE  
 CAN\_TI0R\_STID\_BIT\_POSITION  
 CAN\_TI0R\_EXID\_BIT\_POSITION  
 CAN\_TDL0R\_DATA0\_BIT\_POSITION  
 CAN\_TDL0R\_DATA1\_BIT\_POSITION  
 CAN\_TDL0R\_DATA2\_BIT\_POSITION  
 CAN\_TDL0R\_DATA3\_BIT\_POSITION  
 TSR\_REGISTER\_INDEX  
 RF0R\_REGISTER\_INDEX  
 RF1R\_REGISTER\_INDEX  
 MSR\_REGISTER\_INDEX  
 ESR\_REGISTER\_INDEX  
 CAN\_TSR\_RQCP0\_BIT\_POSITION  
 CAN\_TSR\_RQCP1\_BIT\_POSITION  
 CAN\_TSR\_RQCP2\_BIT\_POSITION  
 CAN\_TSR\_TXOK0\_BIT\_POSITION  
 CAN\_TSR\_TXOK1\_BIT\_POSITION  
 CAN\_TSR\_TXOK2\_BIT\_POSITION  
 CAN\_TSR\_TME0\_BIT\_POSITION

CAN\_TSR\_TME1\_BIT\_POSITION  
CAN\_TSR\_TME2\_BIT\_POSITION  
CAN\_RF0R\_FF0\_BIT\_POSITION  
CAN\_RF0R\_FOV0\_BIT\_POSITION  
CAN\_RF1R\_FF1\_BIT\_POSITION  
CAN\_RF1R\_FOV1\_BIT\_POSITION  
CAN\_MSR\_WKU\_BIT\_POSITION  
CAN\_MSR\_SLAK\_BIT\_POSITION  
CAN\_MSR\_SLAKI\_BIT\_POSITION  
CAN\_ESR\_EWG\_BIT\_POSITION  
CAN\_ESR\_EPV\_BIT\_POSITION  
CAN\_ESR\_BOF\_BIT\_POSITION  
CAN\_FLAG\_MASK  
CAN\_TXMAILBOX\_0  
CAN\_TXMAILBOX\_1  
CAN\_TXMAILBOX\_2

**CAN Private Macros**

IS\_CAN\_MODE  
IS\_CAN\_SJW  
IS\_CAN\_BS1  
IS\_CAN\_BS2  
IS\_CAN\_FILTER\_MODE  
IS\_CAN\_FILTER\_SCALE  
IS\_CAN\_FILTER\_FIFO  
IS\_CAN\_IDTYPE  
IS\_CAN\_RTR  
IS\_CAN\_FIFO  
IS\_CAN\_BANKNUMBER  
IS\_CAN\_TRANSMITMAILBOX  
IS\_CAN\_STID  
IS\_CAN\_EXTID  
IS\_CAN\_DLC  
IS\_CAN\_PRESCALER

**CAN Receive FIFO Number**

CAN\_FIFO0                    CAN FIFO 0 used to receive  
CAN\_FIFO1                    CAN FIFO 1 used to receive

**CAN Remote Transmission Request**

CAN\_RTR\_DATA      Data frame

CAN\_RTR\_REMOTE    Remote frame

***CAN Synchronization Jump Width***

CAN\_SJW\_1TQ      1 time quantum

CAN\_SJW\_2TQ      2 time quantum

CAN\_SJW\_3TQ      3 time quantum

CAN\_SJW\_4TQ      4 time quantum

***CAN Time Quantum in Bit Segment 1***

CAN\_BS1\_1TQ      1 time quantum

CAN\_BS1\_2TQ      2 time quantum

CAN\_BS1\_3TQ      3 time quantum

CAN\_BS1\_4TQ      4 time quantum

CAN\_BS1\_5TQ      5 time quantum

CAN\_BS1\_6TQ      6 time quantum

CAN\_BS1\_7TQ      7 time quantum

CAN\_BS1\_8TQ      8 time quantum

CAN\_BS1\_9TQ      9 time quantum

CAN\_BS1\_10TQ     10 time quantum

CAN\_BS1\_11TQ     11 time quantum

CAN\_BS1\_12TQ     12 time quantum

CAN\_BS1\_13TQ     13 time quantum

CAN\_BS1\_14TQ     14 time quantum

CAN\_BS1\_15TQ     15 time quantum

CAN\_BS1\_16TQ     16 time quantum

***CAN Time Quantum in Bit Segment 2***

CAN\_BS2\_1TQ      1 time quantum

CAN\_BS2\_2TQ      2 time quantum

CAN\_BS2\_3TQ      3 time quantum

CAN\_BS2\_4TQ      4 time quantum

CAN\_BS2\_5TQ      5 time quantum

CAN\_BS2\_6TQ      6 time quantum

CAN\_BS2\_7TQ      7 time quantum

CAN\_BS2\_8TQ      8 time quantum

***CAN Transmit Constants***

CAN\_TXSTATUS\_NOMAILBOX    CAN cell did not provide CAN\_TxStatus\_NoMailBox

## 7 HAL CAN Extension Driver

### 7.1 CANEx Firmware driver registers structures

#### 7.1.1 CAN\_FilterConfTypeDef

**CAN\_FilterConfTypeDef** is defined in the stm32f1xx\_hal\_can\_ex.h

##### Data Fields

- **uint32\_t FilterIdHigh**
- **uint32\_t FilterIdLow**
- **uint32\_t FilterMaskIdHigh**
- **uint32\_t FilterMaskIdLow**
- **uint32\_t FilterFIFOAssignment**
- **uint32\_t FilterNumber**
- **uint32\_t FilterMode**
- **uint32\_t FilterScale**
- **uint32\_t FilterActivation**
- **uint32\_t BankNumber**

##### Field Documentation

- **uint32\_t CAN\_FilterConfTypeDef::FilterIdHigh** Specifies the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- **uint32\_t CAN\_FilterConfTypeDef::FilterIdLow** Specifies the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- **uint32\_t CAN\_FilterConfTypeDef::FilterMaskIdHigh** Specifies the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- **uint32\_t CAN\_FilterConfTypeDef::FilterMaskIdLow** Specifies the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- **uint32\_t CAN\_FilterConfTypeDef::FilterFIFOAssignment** Specifies the FIFO (0 or 1) which will be assigned to the filter. This parameter can be a value of [CAN\\_filter\\_FIFO](#)
- **uint32\_t CAN\_FilterConfTypeDef::FilterNumber** Specifies the filter which will be initialized. This parameter must be a number between Min\_Data = 0 and Max\_Data = 13.
- **uint32\_t CAN\_FilterConfTypeDef::FilterMode** Specifies the filter mode to be initialized. This parameter can be a value of [CAN\\_filter\\_mode](#)
- **uint32\_t CAN\_FilterConfTypeDef::FilterScale** Specifies the filter scale. This parameter can be a value of [CAN\\_filter\\_scale](#)
- **uint32\_t CAN\_FilterConfTypeDef::FilterActivation** Enable or disable the filter. This parameter can be set to ENABLE or DISABLE.
- **uint32\_t CAN\_FilterConfTypeDef::BankNumber** Select the start slave bank filter. This parameter must be a number between Min\_Data = 0 and Max\_Data = 28.



## 7.2 CANEx Firmware driver defines

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

### 7.2.1 CANEx

CANEx

***CAN Extended Private Macros***

IS\_CAN\_FILTER\_NUMBER

## 8 HAL CEC Generic Driver

### 8.1 CEC Firmware driver registers structures

#### 8.1.1 CEC\_InitTypeDef

**CEC\_InitTypeDef** is defined in the stm32f1xx\_hal\_cec.h

##### Data Fields

- **uint32\_t TimingErrorFree**
- **uint32\_t PeriodErrorFree**
- **uint8\_t InitiatorAddress**

##### Field Documentation

- **uint32\_t CEC\_InitTypeDef::TimingErrorFree** Configures the CEC Bit Timing Error Mode. This parameter can be a value of [CEC\\_BitTimingErrorMode](#)
- **uint32\_t CEC\_InitTypeDef::PeriodErrorFree** Configures the CEC Bit Period Error Mode. This parameter can be a value of [CEC\\_BitPeriodErrorMode](#)
- **uint8\_t CEC\_InitTypeDef::InitiatorAddress** Initiator address (source logical address, sent in each header) This parameter can be a value <= 0xF

#### 8.1.2 CEC\_HandleTypeDef

**CEC\_HandleTypeDef** is defined in the stm32f1xx\_hal\_cec.h

##### Data Fields

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

##### Field Documentation

- **CEC\_TypeDef\* CEC\_HandleTypeDef::Instance** CEC registers base address
- **CEC\_InitTypeDef CEC\_HandleTypeDef::Init** CEC communication parameters
- **uint8\_t\* CEC\_HandleTypeDef::pTxBuffPtr** Pointer to CEC Tx transfer Buffer
- **uint16\_t CEC\_HandleTypeDef::TxXferCount** CEC Tx Transfer Counter
- **uint8\_t\* CEC\_HandleTypeDef::pRxBuffPtr** Pointer to CEC Rx transfer Buffer
- **uint16\_t CEC\_HandleTypeDef::RxXferSize** CEC Rx Transfer size, 0: header received only
- **uint32\_t CEC\_HandleTypeDef::ErrorCode** For errors handling purposes, copy of ESR register in case error is reported
- **HAL\_LockTypeDef CEC\_HandleTypeDef::Lock** Locking object

- ***HAL\_CEC\_StateTypeDef CEC\_HandleTypeDef::State*** CEC communication state

## 8.2 CEC Firmware driver API description

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

### 8.2.1 How to use this driver

The CEC HAL driver can be used as follows:

1. Declare a CEC\_HandleTypeDef handle structure.
2. Initialize the CEC low level resources by implementing the HAL\_CEC\_MspInit ()API:
  - a. Enable the CEC interface clock.
  - b. Enable the clock for the CEC GPIOs.
  - c. Configure these CEC pins as alternate function pull-up.
  - d. NVIC configuration if you need to use interrupt process (HAL\_CEC\_Transmit\_IT() and HAL\_CEC\_Receive\_IT() APIs):
  - e. Configure the CEC interrupt priority.
  - f. Enable the NVIC CEC IRQ handle.
  - g. The CEC interrupt is activated/deactivated by the HAL driver
3. Program the Bit Timing Error Mode and the Bit Period Error Mode in the hcec Init structure.
4. Initialize the CEC registers by calling the HAL\_CEC\_Init() API.
5. This API (HAL\_CEC\_Init()) configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL\_CEC\_MspInit() API.

### 8.2.2 Initialization and Configuration functions

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

- The following parameters need to be configured:
  - TimingErrorFree
  - PeriodErrorFree
  - InitiatorAddress
- [\*\*\*HAL\\_CEC\\_Init\(\)\*\*\*](#)
- [\*\*\*HAL\\_CEC\\_DeInit\(\)\*\*\*](#)
- [\*\*\*HAL\\_CEC\\_MspInit\(\)\*\*\*](#)
- [\*\*\*HAL\\_CEC\\_MspDeInit\(\)\*\*\*](#)

### 8.2.3 IO operation functions

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

1. There are two modes of transfer:
  - a. 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.
  - b. No-Blocking mode: The communication is performed using Interrupts. These API's return the HAL status. The end of the data processing will be indicated through the dedicated CEC IRQ when using Interrupt mode. The HAL\_CEC\_TxCpltCallback(), HAL\_CEC\_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The

- HAL\_CEC\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode API's are :
    - a. HAL\_CEC\_Transmit()
    - b. HAL\_CEC\_Receive()
  3. Non-Blocking mode API's with Interrupt are :
    - a. HAL\_CEC\_Transmit\_IT()
    - b. HAL\_CEC\_Receive\_IT()
    - c. HAL\_CEC\_IRQHandler()
  4. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
    - a. HAL\_CEC\_TxCpltCallback()
    - b. HAL\_CEC\_RxCpltCallback()
    - c. HAL\_CEC\_ErrorCallback()
    - [HAL\\_CEC\\_Transmit\(\)](#)
    - [HAL\\_CEC\\_Receive\(\)](#)
    - [HAL\\_CEC\\_Transmit\\_IT\(\)](#)
    - [HAL\\_CEC\\_Receive\\_IT\(\)](#)
    - [HAL\\_CEC\\_GetReceivedFrameSize\(\)](#)
    - [HAL\\_CEC\\_IRQHandler\(\)](#)
    - [HAL\\_CEC\\_TxCpltCallback\(\)](#)
    - [HAL\\_CEC\\_RxCpltCallback\(\)](#)
    - [HAL\\_CEC\\_ErrorCallback\(\)](#)

#### 8.2.4 Peripheral Control functions

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

- HAL\_CEC\_GetState() API can be helpful to check in run-time the state of the CEC peripheral.
- HAL\_CEC\_GetError() API can be helpful to get the error code of a failed transmission or reception.
- [HAL\\_CEC\\_GetState\(\)](#)
- [HAL\\_CEC\\_GetError\(\)](#)

#### 8.2.5 HAL\_CEC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Init (CEC_HandleTypeDef * hcec)</b>
Function Description	Initializes the CEC mode according to the specified parameters in the CEC_InitTypeDef and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 8.2.6 HAL\_CEC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_CEC_DeInit (CEC_HandleTypeDef * hcec)</b>
Function Description	DeInitializes the CEC peripheral.

Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 8.2.7 HAL\_CEC\_MspltInit

Function Name	<b>void HAL_CEC_MspltInit (CEC_HandleTypeDef * hcec)</b>
Function Description	CEC MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 8.2.8 HAL\_CEC\_MspDeInit

Function Name	<b>void HAL_CEC_MspDeInit (CEC_HandleTypeDef * hcec)</b>
Function Description	CEC MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 8.2.9 HAL\_CEC\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Transmit (CEC_HandleTypeDef * hcec, uint8_t DestinationAddress, uint8_t * pData, uint32_t Size, uint32_t Timeout)</b>
Function Description	Send data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> <li>• <b>DestinationAddress</b>: destination logical address</li> <li>• <b>pData</b>: pointer to input byte data buffer</li> <li>• <b>Size</b>: amount of data to be sent in bytes (without counting the header). 0 means only the header is sent (ping operation). Maximum TX size is 15 bytes (1 opcode and up to 14 operands).</li> <li>• <b>Timeout</b>: Timeout duration.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 8.2.10 HAL\_CEC\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Receive (CEC_HandleTypeDef * hcec, uint8_t * pData, uint32_t Timeout)</b>
Function Description	Receive data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> <li>• <b>pData</b>: pointer to received data buffer.</li> <li>• <b>Timeout</b>: Timeout duration.</li> </ul>

Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The received data size is not known beforehand, the latter is known when the reception is complete and is stored in <code>hcec-&gt;RxXferSize</code>. <code>hcec-&gt;RxXferSize</code> is the sum of opcodes + operands (0 to 14 operands max). If only a header is received, <code>hcec-&gt;RxXferSize</code> = 0</li> </ul>

### 8.2.11 HAL\_CEC\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Transmit_IT</b> (CEC_HandleTypeDef * hcec, uint8_t DestinationAddress, uint8_t * pData, uint32_t Size)
Function Description	Send data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> <li>• <b>DestinationAddress</b>: destination logical address</li> <li>• <b>pData</b>: pointer to input byte data buffer</li> <li>• <b>Size</b>: amount of data to be sent in bytes (without counting the header). 0 means only the header is sent (ping operation). Maximum TX size is 15 bytes (1 opcode and up to 14 operands).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 8.2.12 HAL\_CEC\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Receive_IT</b> (CEC_HandleTypeDef * hcec, uint8_t * pData)
Function Description	Receive data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> <li>• <b>pData</b>: pointer to received data buffer.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The received data size is not known beforehand, the latter is known when the reception is complete and is stored in <code>hcec-&gt;RxXferSize</code>. <code>hcec-&gt;RxXferSize</code> is the sum of opcodes + operands (0 to 14 operands max). If only a header is received, <code>hcec-&gt;RxXferSize</code> = 0</li> </ul>

### 8.2.13 HAL\_CEC\_GetReceivedFrameSize

Function Name	<b>uint32_t HAL_CEC_GetReceivedFrameSize</b> (CEC_HandleTypeDef * hcec)
Function Description	Get size of the received frame.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Frame size</li> </ul>

## 8.2.14 HAL\_CEC\_IRQHandler

Function Name	<b>void HAL_CEC_IRQHandler (CEC_HandleTypeDef * hcec)</b>
Function Description	This function handles CEC interrupt requests.
Parameters	<ul style="list-style-type: none"><li>• <b>hcec</b>: CEC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 8.2.15 HAL\_CEC\_TxCpltCallback

Function Name	<b>void HAL_CEC_TxCpltCallback (CEC_HandleTypeDef * hcec)</b>
Function Description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hcec</b>: CEC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 8.2.16 HAL\_CEC\_RxCpltCallback

Function Name	<b>void HAL_CEC_RxCpltCallback (CEC_HandleTypeDef * hcec)</b>
Function Description	Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hcec</b>: CEC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 8.2.17 HAL\_CEC\_ErrorCallback

Function Name	<b>void HAL_CEC_ErrorCallback (CEC_HandleTypeDef * hcec)</b>
Function Description	CEC error callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hcec</b>: CEC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 8.2.18 HAL\_CEC\_GetState

Function Name	<b>HAL_CEC_StateTypeDef HAL_CEC_GetState (CEC_HandleTypeDef * hcec)</b>
Function Description	return the CEC state
Parameters	<ul style="list-style-type: none"><li>• <b>hcec</b>: CEC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL state</li></ul>

## 8.2.19 HAL\_CEC\_GetError

Function Name	<b>uint32_t HAL_CEC_GetError (CEC_HandleTypeDef * hcec)</b>
Function Description	Return the CEC error code.
Parameters	<ul style="list-style-type: none"> <li><b>hcec</b>: : pointer to a CEC_HandleTypeDef structure that contains the configuration information for the specified CEC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>CEC Error Code</li> </ul>

## 8.3 CEC Firmware driver defines

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

### 8.3.1 CEC

CEC

#### **Bit Period Error Mode**

CEC\_BIT\_PERIOD\_ERROR\_MODE\_STANDARD Bit period error Standard Mode

CEC\_BIT\_PERIOD\_ERROR\_MODE\_FLEXIBLE Bit period error Flexible Mode

#### **Bit Timing Error Mode**

CEC\_BIT\_TIMING\_ERROR\_MODE\_STANDARD Bit timing error Standard Mode

CEC\_BIT\_TIMING\_ERROR\_MODE\_ERRORFREE Bit timing error Free Mode

#### **CEC Exported Macros**

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

#### **Description:**

- Reset CEC handle state.

#### **Parameters:**

- \_\_HANDLE\_\_**: CEC handle.

#### **Return value:**

- None:

**\_\_HAL\_CEC\_GET\_FLAG**

#### **Description:**

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

#### **Parameters:**

- \_\_HANDLE\_\_**: specifies the CEC Handle.
- \_\_INTERRUPT\_\_**: specifies the interrupt to check.
  - CEC\_FLAG\_TERR: Tx Error
  - CEC\_FLAG\_TBTF: Tx Block Transfer Finished
  - CEC\_FLAG\_RERR: Rx Error
  - CEC\_FLAG\_RBTF: Rx Block Transfer Finished



**\_\_HAL\_CEC\_CLEAR\_FLAG****Return value:**

- ITStatus:

**Description:**

- Clears the CEC's pending flags.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the CEC Handle.
- **\_\_FLAG\_\_**: specifies the flag to clear. This parameter can be any combination of the following values:
  - CEC\_CSR\_TERR: Tx Error
  - CEC\_CSR\_TBTF: Tx Block Transfer Finished
  - CEC\_CSR\_RERR: Rx Error
  - CEC\_CSR\_RBTF: Rx Block Transfer Finished

**Return value:**

- none:

**Description:**

- Enables the specified CEC interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the CEC Handle.
- **\_\_INTERRUPT\_\_**: The CEC interrupt to enable. This parameter can be:
  - CEC\_IT\_IE : Interrupt Enable

**Return value:**

- none:

**Description:**

- Disables the specified CEC interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the CEC Handle.
- **\_\_INTERRUPT\_\_**: The CEC interrupt to enable. This parameter can be:
  - CEC\_IT\_IE : Interrupt

**\_\_HAL\_CEC\_ENABLE\_IT****\_\_HAL\_CEC\_DISABLE\_IT**

Enable

`__HAL_CEC_GET_IT_SOURCE`**Return value:**

- none:

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.
- `__INTERRUPT__`: The CEC interrupt to enable. This parameter can be:
  - `CEC_IT_IE` : Interrupt Enable

**Return value:**

- FlagStatus:

**Description:**

- Enables the CEC device.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none:

**Description:**

- Disables the CEC device.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none:

**Description:**

- Set Transmission Start flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none:

**Description:**`__HAL_CEC_ENABLE``__HAL_CEC_DISABLE``__HAL_CEC_FIRST_BYTE_TX_SET``__HAL_CEC_LAST_BYTE_TX_SET`

- Set Transmission End flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none:

`__HAL_CEC_GET_TRANSMISSION_START_FLAG`**Description:**

- Get Transmission Start flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- FlagStatus:

`__HAL_CEC_GET_TRANSMISSION_END_FLAG`**Description:**

- Get Transmission End flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- FlagStatus:

`__HAL_CEC_CLEAR_OAR`**Description:**

- Clear OAR register.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none:

`__HAL_CEC_SET_OAR`**Description:**

- Set OAR register.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.
- `__ADDRESS__`: Own Address value.

**Return value:**

- none:

**Flags definition**`CEC_FLAG_TSOM`

CEC\_FLAG\_TEOM

CEC\_FLAG\_TERR

CEC\_FLAG\_TBTRF

CEC\_FLAG\_RSOM

CEC\_FLAG\_REOM

CEC\_FLAG\_RERR

CEC\_FLAG\_RBTF

**Initiator logical address position in message header**

CEC\_INITIATOR\_LSB\_POS

**Interrupts definition**

CEC\_IT\_IE

**CEC Private Constants**

CEC\_CFGR\_FIELDS

CEC\_FLAG\_TRANSMIT\_MASK

CEC\_FLAG\_RECEIVE\_MASK

CEC\_ESR\_ALL\_ERROR

CEC\_RXXFERSIZE\_INITIALIZE

Value used to initialise the RxXferSize of the handle

IS\_CEC\_BIT\_TIMING\_ERROR\_MODE

IS\_CEC\_BIT\_PERIOD\_ERROR\_MODE

IS\_CEC\_OAR\_ADDRESS

**Description:**

- Check CEC device Own Address Register (OAR) setting.

**Parameters:**

- `__ADDRESS__`: CEC own address.

**Return value:**

- Test: result (TRUE or FALSE).

IS\_CEC\_ADDRESS

**Description:**

- Check CEC initiator or destination logical address setting.

**Parameters:**

- `__ADDRESS__`: CEC initiator or logical address.

**Return value:**

- Test: result (TRUE or FALSE).

IS\_CEC\_MSGSIZE

**Description:**

- Check CEC message size.

**Parameters:**

- `__SIZE__`: CEC message size.

**Return value:**

- Test: result (TRUE or FALSE).

## 9 HAL CORTEX Generic Driver

### 9.1 CORTEX Firmware driver API description

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

#### 9.1.1 Initialization and de-initialization functions

This section provide the Cortex HAL driver functions allowing to configure Interrupts SysTick functionalities

- [\*HAL\\_NVIC\\_SetPriorityGrouping\(\)\*](#)
- [\*HAL\\_NVIC\\_SetPriority\(\)\*](#)
- [\*HAL\\_NVIC\\_EnableIRQ\(\)\*](#)
- [\*HAL\\_NVIC\\_DisableIRQ\(\)\*](#)
- [\*HAL\\_NVIC\\_SystemReset\(\)\*](#)
- [\*HAL\\_SYSTICK\\_Config\(\)\*](#)

#### 9.1.2 Peripheral Control functions

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

- [\*HAL\\_NVIC\\_GetPriorityGrouping\(\)\*](#)
- [\*HAL\\_NVIC\\_GetPriority\(\)\*](#)
- [\*HAL\\_NVIC\\_SetPendingIRQ\(\)\*](#)
- [\*HAL\\_NVIC\\_GetPendingIRQ\(\)\*](#)
- [\*HAL\\_NVIC\\_ClearPendingIRQ\(\)\*](#)
- [\*HAL\\_NVIC\\_GetActive\(\)\*](#)
- [\*HAL\\_SYSTICK\\_CLKSourceConfig\(\)\*](#)
- [\*HAL\\_SYSTICK\\_IRQHandler\(\)\*](#)
- [\*HAL\\_SYSTICK\\_Callback\(\)\*](#)

#### 9.1.3 HAL\_NVIC\_SetPriorityGrouping

Function Name	<b>void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)</b>
Function Description	Sets the priority grouping field (pre-emption priority and subpriority) using the required unlock sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>PriorityGroup:</b> The priority grouping bits length. This parameter can be one of the following values:            NVIC_PRIORITYGROUP_0: 0 bits for pre-emption priority 4 bits for subpriority            NVIC_PRIORITYGROUP_1: 1 bits for pre-emption priority 3 bits for subpriority            NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority 2 bits for subpriority            NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority 1 bits for subpriority            NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority 0 bits for subpriority         </li> </ul>

Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the NVIC_PriorityGroup_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the subpriority.</li> </ul>

### 9.1.4 HAL\_NVIC\_SetPriority

Function Name	<b>void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)</b>
Function Description	Sets the priority of an interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn:</b> External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))</li> <li>• <b>PreemptPriority:</b> The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority</li> <li>• <b>SubPriority:</b> the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 9.1.5 HAL\_NVIC\_EnableIRQ

Function Name	<b>void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)</b>
Function Description	Enables a device specific interrupt in the NVIC interrupt controller.
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn:</b> External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.</li> </ul>

### 9.1.6 HAL\_NVIC\_DisableIRQ

Function Name	<b>void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)</b>
Function Description	Disables a device specific interrupt in the NVIC interrupt controller.
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn:</b> External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 9.1.7 HAL\_NVIC\_SystemReset

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

### 9.1.8 HAL\_SYSTICK\_Config

Function Name	<b>uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)</b>
Function Description	Initializes the System Timer and its interrupt, and starts the System Tick Timer.
Parameters	<ul style="list-style-type: none"> <li>• <b>TicksNumb:</b> Specifies the ticks Number of ticks between two interrupts.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• status - 0 Function succeeded. 1 Function failed.</li> </ul>

### 9.1.9 HAL\_NVIC\_GetPriorityGrouping

Function Name	<b>uint32_t HAL_NVIC_GetPriorityGrouping (void )</b>
Function Description	Gets the priority grouping field from the NVIC Interrupt Controller.
Return values	<ul style="list-style-type: none"> <li>• Priority grouping field (SCB-&gt;AIRCR [10:8] PRIGROUP field)</li> </ul>

### 9.1.10 HAL\_NVIC\_GetPriority

Function Name	<b>void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority, uint32_t * pSubPriority)</b>
Function Description	Gets the priority of an interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn:</b> External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))</li> <li>• <b>PriorityGroup:</b> the priority grouping bits length. This parameter can be one of the following values:            NVIC_PRIORITYGROUP_0: 0 bits for pre-emption priority 4 bits for subpriority            NVIC_PRIORITYGROUP_1: 1 bits for pre-emption priority 3 bits for subpriority            NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority 2 bits for subpriority            NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority 1 bits for subpriority            NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority 0 bits for subpriority</li> <li>• <b>pPreemptPriority:</b> Pointer on the Preemptive priority value (starting from 0).</li> <li>• <b>pSubPriority:</b> Pointer on the Subpriority value (starting from</li> </ul>



0).

Return values

- None

### 9.1.11 HAL\_NVIC\_SetPendingIRQ

Function Name **void HAL\_NVIC\_SetPendingIRQ (IRQn\_Type IRQn)**

Function Description Sets Pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn\_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))

Return values

- None

### 9.1.12 HAL\_NVIC\_GetPendingIRQ

Function Name **uint32\_t HAL\_NVIC\_GetPendingIRQ (IRQn\_Type IRQn)**

Function Description Gets Pending Interrupt (reads the pending register in the NVIC and returns the pending bit for the specified interrupt).

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn\_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))

Return values

- status - 0 Interrupt status is not pending. 1 Interrupt status is pending.

### 9.1.13 HAL\_NVIC\_ClearPendingIRQ

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

Function Description Clears the pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn\_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))

Return values

- None

### 9.1.14 HAL\_NVIC\_GetActive

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

Function Description Gets active interrupt ( reads the active register in NVIC and returns the active bit).

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn\_Type enumeration (For the complete

STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f10xxx.h))

- Return values
- status - 0 Interrupt status is not pending. 1 Interrupt status is pending.

### 9.1.15 HAL\_SYSTICK\_CLKSourceConfig

- Function Name      **void HAL\_SYSTICK\_CLKSourceConfig (uint32\_t CLKSource)**
- Function Description    Configures the SysTick clock source.
- Parameters
- **CLKSource:** specifies the SysTick clock source. This parameter can be one of the following values:  
SYSTICK\_CLKSOURCE\_HCLK\_DIV8: AHB clock divided by 8 selected as SysTick clock source.  
SYSTICK\_CLKSOURCE\_HCLK: AHB clock selected as SysTick clock source.
- Return values
- None

### 9.1.16 HAL\_SYSTICK\_IRQHandler

- Function Name      **void HAL\_SYSTICK\_IRQHandler (void )**
- Function Description    This function handles SYSTICK interrupt request.
- Return values
- None

### 9.1.17 HAL\_SYSTICK\_Callback

- Function Name      **void HAL\_SYSTICK\_Callback (void )**
- Function Description    SYSTICK callback.
- Return values
- None

## 9.2 CORTEX Firmware driver defines

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

### 9.2.1 CORTEX

CORTEX

#### ***CORTEX Preemption Priority Group***

- NVIC\_PRIORITYGROUP\_0    0 bits for pre-emption priority 4 bits for subpriority
- NVIC\_PRIORITYGROUP\_1    1 bits for pre-emption priority 3 bits for subpriority
- NVIC\_PRIORITYGROUP\_2    2 bits for pre-emption priority 2 bits for subpriority
- NVIC\_PRIORITYGROUP\_3    3 bits for pre-emption priority 1 bits for subpriority
- NVIC\_PRIORITYGROUP\_4    4 bits for pre-emption priority 0 bits for subpriority

**CORTEX Preemption Priority Group**

IS\_NVIC\_PRIORITY\_GROUP

IS\_NVIC\_PREEMPTION\_PRIORITY

IS\_NVIC\_SUB\_PRIORITY

IS\_NVIC\_DEVICE\_IRQ

**CORTEX SysTick clock source**

SYSTICK\_CLKSOURCE\_HCLK\_DIV8

SYSTICK\_CLKSOURCE\_HCLK

**CORTEX SysTick clock source**

\_\_HAL\_CORTEX\_SYSTICKCLK\_CONFIG

**Description:**

- Configures the SysTick clock source.

**Parameters:**

- `__CLKSRC__`: specifies the SysTick clock source. This parameter can be one of the following values:
  - SYSTICK\_CLKSOURCE\_HCLK\_DIV8: AHB clock divided by 8 selected as SysTick clock source.
  - SYSTICK\_CLKSOURCE\_HCLK: AHB clock selected as SysTick clock source.

**Return value:**

- None:

**CORTEX SysTick clock source**

IS\_SYSTICK\_CLK\_SOURCE

## 10 HAL CRC Generic Driver

### 10.1 CRC Firmware driver registers structures

#### 10.1.1 CRC\_HandleTypeDef

*CRC\_HandleTypeDef* is defined in the stm32f1xx\_hal\_crc.h

##### Data Fields

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

##### Field Documentation

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

### 10.2 CRC Firmware driver API description

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

#### 10.2.1 How to use this driver

The CRC HAL driver can be used as follows:

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

#### 10.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRC according to the specified parameters in the `CRC_InitTypeDef` and create the associated handle
- DeInitialize the CRC peripheral
- Initialize the CRC MSP
- DeInitialize CRC MSP
- [\*HAL\\_CRC\\_Init\(\)\*](#)
- [\*HAL\\_CRC\\_DeInit\(\)\*](#)
- [\*HAL\\_CRC\\_MspInit\(\)\*](#)
- [\*HAL\\_CRC\\_MspDeInit\(\)\*](#)

## 10.2.3 Peripheral Control functions

This section provides functions allowing to:

- Compute the 32-bit CRC value of 32-bit data buffer, using combination of the previous CRC value and the new one.
- Compute the 32-bit CRC value of 32-bit data buffer, independently of the previous CRC value.
- [HAL\\_CRC\\_Accumulate\(\)](#)
- [HAL\\_CRC\\_Calculate\(\)](#)

## 10.2.4 Peripheral State functions

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

- [HAL\\_CRC\\_GetState\(\)](#)

## 10.2.5 HAL\_CRC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)</b>
Function Description	Initializes the CRC according to the specified parameters in the CRC_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc</b>: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 10.2.6 HAL\_CRC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)</b>
Function Description	DeInitializes the CRC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc</b>: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 10.2.7 HAL\_CRC\_MspInit

Function Name	<b>void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)</b>
Function Description	Initializes the CRC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc</b>: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 10.2.8 HAL\_CRC\_MspDeInit

Function Name	<b>void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)</b>
Function Description	DeInitializes the CRC MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hcrc</b>: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 10.2.9 HAL\_CRC\_Accumulate

Function Name	<b>uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)</b>
Function Description	Computes the 32-bit CRC of 32-bit data buffer using combination of the previous CRC value and the new one.
Parameters	<ul style="list-style-type: none"><li>• <b>hcrc</b>: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC</li><li>• <b>pBuffer</b>: pointer to the buffer containing the data to be computed</li><li>• <b>BufferLength</b>: length of the buffer to be computed (defined in word, 4 bytes)</li></ul>
Return values	<ul style="list-style-type: none"><li>• 32-bit CRC</li></ul>

## 10.2.10 HAL\_CRC\_Calculate

Function Name	<b>uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)</b>
Function Description	Computes the 32-bit CRC of 32-bit data buffer independently of the previous CRC value.
Parameters	<ul style="list-style-type: none"><li>• <b>hcrc</b>: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC</li><li>• <b>pBuffer</b>: Pointer to the buffer containing the data to be computed</li><li>• <b>BufferLength</b>: Length of the buffer to be computed (defined in word, 4 bytes)</li></ul>
Return values	<ul style="list-style-type: none"><li>• 32-bit CRC</li></ul>

## 10.2.11 HAL\_CRC\_GetState

Function Name	<b>HAL_CRC_StateTypeDef HAL_CRC_GetState (CRC_HandleTypeDef * hcrc)</b>
Function Description	Returns the CRC state.
Parameters	<ul style="list-style-type: none"><li>• <b>hcrc</b>: pointer to a CRC_HandleTypeDef structure that</li></ul>

contains the configuration information for CRC

Return values

- HAL state

## 10.3 CRC Firmware driver defines

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

### 10.3.1 CRC

CRC

#### **CRC Exported Macros**

`__HAL_CRC_RESET_HANDLE_STATE`

#### **Description:**

- Reset CRC handle state.

#### **Parameters:**

- `__HANDLE__`: CRC handle

#### **Return value:**

- None:

`__HAL_CRC_DR_RESET`

#### **Description:**

- Resets CRC Data Register.

#### **Parameters:**

- `__HANDLE__`: CRC handle

#### **Return value:**

- None:

`__HAL_CRC_SET_IDR`

#### **Description:**

- Stores a 8-bit data in the Independent Data(ID) register.

#### **Parameters:**

- `__HANDLE__`: CRC handle
- `__VALUE__`: 8-bit value to be stored in the ID register

#### **Return value:**

- None:

`__HAL_CRC_GET_IDR`

#### **Description:**

- Returns the 8-bit data stored in the Independent Data(ID) register.

#### **Parameters:**

- `__HANDLE__`: CRC handle

#### **Return value:**

- 8-bit: value of the ID register

## 11 HAL DAC Generic Driver

### 11.1 DAC Firmware driver registers structures

#### 11.1.1 DAC\_HandleTypeDef

*DAC\_HandleTypeDef* is defined in the stm32f1xx\_hal\_dac.h

##### Data Fields

- *DAC\_TypeDef \* Instance*
- *\_\_IO HAL\_DAC\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*
- *DMA\_HandleTypeDef \* DMA\_Handle1*
- *DMA\_HandleTypeDef \* DMA\_Handle2*
- *\_\_IO uint32\_t ErrorCode*

##### Field Documentation

- *DAC\_TypeDef\* DAC\_HandleTypeDef::Instance* Register base address
- *\_\_IO HAL\_DAC\_StateTypeDef DAC\_HandleTypeDef::State* DAC communication state
- *HAL\_LockTypeDef DAC\_HandleTypeDef::Lock* DAC locking object
- *DMA\_HandleTypeDef\* DAC\_HandleTypeDef::DMA\_Handle1* Pointer DMA handler for channel 1
- *DMA\_HandleTypeDef\* DAC\_HandleTypeDef::DMA\_Handle2* Pointer DMA handler for channel 2
- *\_\_IO uint32\_t DAC\_HandleTypeDef::ErrorCode* DAC Error code

#### 11.1.2 DAC\_ChannelConfTypeDef

*DAC\_ChannelConfTypeDef* is defined in the stm32f1xx\_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 [DACEx\\_trigger\\_selection](#) Note: For STM32F100x high-density value line devices, additional trigger sources are available.
- *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](#)



## 11.2 DAC Firmware driver API description

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

### 11.2.1 DAC Peripheral features

#### DAC Channels

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

1. DAC channel1 with DAC\_OUT1 (PA4) as output
2. DAC channel2 with DAC\_OUT2 (PA5) as output

#### DAC Triggers

Digital to Analog conversion can be non-triggered using DAC\_TRIGGER\_NONE and DAC\_OUT1/DAC\_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx\_PIN\_9) using DAC\_TRIGGER\_EXT\_IT9. The used pin (GPIOx\_PIN\_9) must be configured in input mode.
2. Timers TRGO: TIM2, TIM4, TIM6, TIM7 For STM32F10x connectivity line devices and STM32F100x devices: TIM3 For STM32F10x high-density and XL-density devices: TIM8 For STM32F100x high-density value line devices: TIM15 as replacement of TIM5. (DAC\_TRIGGER\_T2\_TRGO, DAC\_TRIGGER\_T4\_TRGO...)
3. Software using DAC\_TRIGGER\_SOFTWARE

#### DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use `sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE`;



Refer to the device datasheet for more details about output impedance value with and without output buffer.

#### DAC connect feature

Each DAC channel can be connected internally. To connect, use `sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_ENABLE`;

#### GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel1 on PA5 is configured in Analog and disabled). Channel1 may disturb channel2 as coupling effect. Note that there is no coupling on channel2 as soon as channel2 is turned on. Coupling on adjacent channel could be avoided as follows: when unused PA5 is configured as INPUT PULL-UP or DOWN. PA5 is configured in ANALOG just before it is turned on.

### DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave using HAL\_DACEx\_NoiseWaveGenerate()
2. Triangle wave using HAL\_DACEx\_TriangleWaveGenerate()

### DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC\_ALIGN\_8B\_R
2. 12-bit left alignment using DAC\_ALIGN\_12B\_L
3. 12-bit right alignment using DAC\_ALIGN\_12B\_R

### DAC data value to voltage correspondance

The analog output voltage on each DAC channel pin is determined by the following equation:

$$\text{DAC\_OUT}_x = \text{VREF+} * \text{DOR} / 4095$$

- with DOR is the Data Output Register

VEF+ is the input voltage reference (refer to the device datasheet)

e.g. To set DAC\_OUT1 to 0.7V, use

- Assuming that VREF+ = 3.3V,  $\text{DAC\_OUT1} = (3.3 * 868) / 4095 = 0.7\text{V}$

### DMA requests

A DMA1 request can be generated when an external trigger (but not a software trigger) occurs if DMA1 requests are enabled using HAL\_DAC\_Start\_DMA()

DMA requests are mapped as following:

1. DAC channel1 : For STM32F100x low-density, medium-density, high-density with DAC DMA remap: mapped on DMA1 channel3 which must be already configured For STM32F100x high-density without DAC DMA remap and other STM32F1 devices: mapped on DMA2 channel3 which must be already configured
2. DAC channel2 : For STM32F100x low-density, medium-density, high-density with DAC DMA remap: mapped on DMA1 channel4 which must be already configured For STM32F100x high-density without DAC DMA remap and other STM32F1 devices: mapped on DMA2 channel4 which must be already configured

## 11.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL\_DAC\_Init()
- Configure DAC\_OUTx (DAC\_OUT1: PA4, DAC\_OUT2: PA5) in analog mode.
- Configure the DAC channel using HAL\_DAC\_ConfigChannel() function.
- Enable the DAC channel using HAL\_DAC\_Start() or HAL\_DAC\_Start\_DMA functions

### Polling mode IO operation

- Start the DAC peripheral using HAL\_DAC\_Start()

- To read the DAC last data output value, use the HAL\_DAC\_GetValue() function.
- Stop the DAC peripheral using HAL\_DAC\_Stop()

### DMA mode IO operation

- Start the DAC peripheral using HAL\_DAC\_Start\_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion
- At the middle of data transfer HAL\_DACEx\_ConvHalfCpltCallbackCh1() or HAL\_DACEx\_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL\_DAC\_ConvHalfCpltCallbackCh1 or HAL\_DAC\_ConvHalfCpltCallbackCh2
- At The end of data transfer HAL\_DAC\_ConvCpltCallbackCh1() or HAL\_DAC\_ConvCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL\_DAC\_ConvCpltCallbackCh1 or HAL\_DAC\_ConvCpltCallbackCh2
- In case of transfer Error, HAL\_DAC\_ErrorCallbackCh1() or HAL\_DACEx\_ErrorCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL\_DAC\_ErrorCallbackCh1 or HAL\_DACEx\_ErrorCallbackCh2
- For STM32F100x devices with specific feature: DMA underrun. In case of DMA underrun, DAC interruption triggers and execute internal function HAL\_DAC\_IRQHandler. HAL\_DAC\_DMAUnderrunCallbackCh1() or HAL\_DACEx\_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL\_DAC\_DMAUnderrunCallbackCh1 or HAL\_DACEx\_DMAUnderrunCallbackCh2 add his own code by customization of function pointer HAL\_DAC\_ErrorCallbackCh1
- Stop the DAC peripheral using HAL\_DAC\_Stop\_DMA()

### DAC HAL driver macros list

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

- `__HAL_DAC_ENABLE` : Enable the DAC peripheral (For STM32F100x devices with specific feature: DMA underrun)
- `__HAL_DAC_DISABLE` : Disable the DAC peripheral (For STM32F100x devices with specific feature: DMA underrun)
- `__HAL_DAC_CLEAR_FLAG`: Clear the DAC's pending flags (For STM32F100x devices with specific feature: DMA underrun)
- `__HAL_DAC_GET_FLAG`: Get the selected DAC's flag status (For STM32F100x devices with specific feature: DMA underrun)



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

## 11.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DAC.
- De-initialize the DAC.
- [\*\*HAL\\_DAC\\_Init\(\)\*\*](#)

- [HAL\\_DAC\\_DeInit\(\)](#)
- [HAL\\_DAC\\_MspInit\(\)](#)
- [HAL\\_DAC\\_MspDeInit\(\)](#)

#### 11.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- [HAL\\_DAC\\_Start\(\)](#)
- [HAL\\_DAC\\_Stop\(\)](#)
- [HAL\\_DAC\\_Start\\_DMA\(\)](#)
- [HAL\\_DAC\\_Stop\\_DMA\(\)](#)
- [HAL\\_DAC\\_GetValue\(\)](#)
- [HAL\\_DAC\\_ConvCpltCallbackCh1\(\)](#)
- [HAL\\_DAC\\_ConvHalfCpltCallbackCh1\(\)](#)
- [HAL\\_DAC\\_ErrorCallbackCh1\(\)](#)
- [HAL\\_DAC\\_SetValue\(\)](#)

#### 11.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.
- [HAL\\_DAC\\_ConfigChannel\(\)](#)
- [HAL\\_DAC\\_SetValue\(\)](#)

#### 11.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.
- [HAL\\_DAC\\_GetState\(\)](#)
- [HAL\\_DAC\\_GetError\(\)](#)
- [HAL\\_DAC\\_ConvCpltCallbackCh1\(\)](#)
- [HAL\\_DAC\\_ConvHalfCpltCallbackCh1\(\)](#)
- [HAL\\_DAC\\_ErrorCallbackCh1\(\)](#)

#### 11.2.7 HAL\_DAC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef *hdac)</b>
Function Description	Initializes the DAC peripheral according to the specified parameters in the DAC_InitStruct.

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

### 11.2.8 HAL\_DAC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)</b>
Function Description	Deinitializes the DAC peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li><b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 11.2.9 HAL\_DAC\_Msplnit

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

### 11.2.10 HAL\_DAC\_MspDeInit

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

### 11.2.11 HAL\_DAC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)</b>
Function Description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none"> <li><b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li><b>Channel:</b> The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selected DAC_CHANNEL_2: DAC Channel2 selected</li> </ul>

Return values

- HAL status

### 11.2.12 HAL\_DAC\_Stop

**Function Name** `HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)`

**Function Description** Disables DAC and stop conversion of channel.

**Parameters**

- **hdac**: pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel**: The selected DAC channel. This parameter can be one of the following values: DAC\_CHANNEL\_1: DAC Channel1 selected DAC\_CHANNEL\_2: DAC Channel2 selected

Return values

- HAL status

### 11.2.13 HAL\_DAC\_Start\_DMA

**Function Name** `HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)`

**Function Description** Enables DAC and starts conversion of channel.

**Parameters**

- **hdac**: pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel**: The selected DAC channel. This parameter can be one of the following values: DAC\_CHANNEL\_1: DAC Channel1 selected DAC\_CHANNEL\_2: DAC Channel2 selected
- **pData**: The destination peripheral Buffer address.
- **Length**: The length of data to be transferred from memory to DAC peripheral
- **Alignment**: Specifies the data alignment for DAC channel. This parameter can be one of the following values:  
DAC\_ALIGN\_8B\_R: 8bit right data alignment selected  
DAC\_ALIGN\_12B\_L: 12bit left data alignment selected  
DAC\_ALIGN\_12B\_R: 12bit right data alignment selected

Return values

- HAL status

### 11.2.14 HAL\_DAC\_Stop\_DMA

**Function Name** `HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)`

**Function Description** Disables DAC and stop conversion of channel.

**Parameters**

- **hdac**: pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel**: The selected DAC channel. This parameter can be one of the following values: DAC\_CHANNEL\_1: DAC

Channel1 selected DAC\_CHANNEL\_2: DAC Channel2 selected

Return values

- HAL status

### 11.2.15 HAL\_DAC\_GetValue

Function Name **uint32\_t HAL\_DAC\_GetValue (DAC\_HandleTypeDef \* hdac, uint32\_t Channel)**

Function Description Returns the last data output value of the selected DAC channel.

Parameters

- **hdac**: pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel**: The selected DAC channel. This parameter can be one of the following values: DAC\_CHANNEL\_1: DAC Channel1 selected DAC\_CHANNEL\_2: DAC Channel2 selected

Return values

- The selected DAC channel data output value.

### 11.2.16 HAL\_DAC\_ConvCpltCallbackCh1

Function Name **void HAL\_DAC\_ConvCpltCallbackCh1 (DAC\_HandleTypeDef \* hdac)**

Function Description Conversion complete callback in non blocking mode for Channel1.

Parameters

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

Return values

- None

### 11.2.17 HAL\_DAC\_ConvHalfCpltCallbackCh1

Function Name **void HAL\_DAC\_ConvHalfCpltCallbackCh1 (DAC\_HandleTypeDef \* hdac)**

Function Description Conversion half DMA transfer callback in non blocking mode for Channel1.

Parameters

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

Return values

- None

### 11.2.18 HAL\_DAC\_ErrorCallbackCh1

Function Name **void HAL\_DAC\_ErrorCallbackCh1 (DAC\_HandleTypeDef \* hdac)**

Function Description Error DAC callback for Channel1.

Parameters

- **hdac**: pointer to a DAC\_HandleTypeDef structure that

contains the configuration information for the specified DAC.

Return values

- None

### 11.2.19 HAL\_DAC\_SetValue

**Function Name** **HAL\_StatusTypeDef HAL\_DAC\_SetValue**  
(DAC\_HandleTypeDef \* hdac, uint32\_t Channel, uint32\_t Alignment, uint32\_t Data)

**Function Description** Set the specified data holding register value for DAC channel.

**Parameters**

- **hdac**: pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel**: The selected DAC channel. This parameter can be one of the following values: DAC\_CHANNEL\_1: DAC Channel1 selected DAC\_CHANNEL\_2: DAC Channel2 selected
- **Alignment**: Specifies the data alignment. This parameter can be one of the following values: DAC\_ALIGN\_8B\_R: 8bit right data alignment selected DAC\_ALIGN\_12B\_L: 12bit left data alignment selected DAC\_ALIGN\_12B\_R: 12bit right data alignment selected
- **Data**: Data to be loaded in the selected data holding register.

**Return values**

- HAL status

### 11.2.20 HAL\_DAC\_ConfigChannel

**Function Name** **HAL\_StatusTypeDef HAL\_DAC\_ConfigChannel**  
(DAC\_HandleTypeDef \* hdac, DAC\_ChannelConfTypeDef \* sConfig, uint32\_t Channel)

**Function Description** Configures the selected DAC channel.

**Parameters**

- **hdac**: pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig**: DAC configuration structure.
- **Channel**: The selected DAC channel. This parameter can be one of the following values: DAC\_CHANNEL\_1: DAC Channel1 selected DAC\_CHANNEL\_2: DAC Channel2 selected

**Return values**

- HAL status

### 11.2.21 HAL\_DAC\_SetValue

**Function Name** **HAL\_StatusTypeDef HAL\_DAC\_SetValue**  
(DAC\_HandleTypeDef \* hdac, uint32\_t Channel, uint32\_t Alignment, uint32\_t Data)

**Function Description** Set the specified data holding register value for DAC channel.

**Parameters**

- **hdac**: pointer to a DAC\_HandleTypeDef structure that



- contains the configuration information for the specified DAC.
  - **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC\_CHANNEL\_1: DAC Channel1 selected DAC\_CHANNEL\_2: DAC Channel2 selected
  - **Alignment:** Specifies the data alignment. This parameter can be one of the following values: DAC\_ALIGN\_8B\_R: 8bit right data alignment selected DAC\_ALIGN\_12B\_L: 12bit left data alignment selected DAC\_ALIGN\_12B\_R: 12bit right data alignment selected
  - **Data:** Data to be loaded in the selected data holding register.
- Return values
- HAL status

### 11.2.22 HAL\_DAC\_GetState

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

### 11.2.23 HAL\_DAC\_GetError

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

### 11.2.24 HAL\_DAC\_ConvCpltCallbackCh1

Function Name	<b>void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)</b>
Function Description	Conversion complete callback in non blocking mode for Channel1.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 11.2.25 HAL\_DAC\_ConvHalfCpltCallbackCh1

Function Name	<b>void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)</b>
---------------	--

Function Description	Conversion half DMA transfer callback in non blocking mode for Channel1.
Parameters	<ul style="list-style-type: none"> <li><b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 11.2.26 HAL\_DAC\_ErrorCallbackCh1

Function Name	<b>void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef *hdac)</b>
Function Description	Error DAC callback for Channel1.
Parameters	<ul style="list-style-type: none"> <li><b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

## 11.3 DAC Firmware driver defines

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

### 11.3.1 DAC

DAC

#### **DAC Channel selection**

DAC\_CHANNEL\_1

DAC\_CHANNEL\_2

#### **DAC data alignement**

DAC\_ALIGN\_12B\_R

DAC\_ALIGN\_12B\_L

DAC\_ALIGN\_8B\_R

#### **DAC Error Code**

HAL\_DAC\_ERROR\_NONE No error

HAL\_DAC\_ERROR\_DMAUNDERRUNCH1 DAC channel1 DMA underrun error

HAL\_DAC\_ERROR\_DMAUNDERRUNCH2 DAC channel2 DMA underrun error

HAL\_DAC\_ERROR\_DMA DMA error

#### **DAC Exported Macros**

\_\_HAL\_DAC\_RESET\_HANDLE\_STATE **Description:**

- Reset DAC handle state.

#### **Parameters:**

- \_\_HANDLE\_\_: specifies the DAC handle.

#### **Return value:**

\_\_HAL\_DAC\_ENABLE

- None:

**Description:**

- Enable the DAC channel.

**Parameters:**

- \_\_HANDLE\_\_: specifies the DAC handle.
- \_\_DAC\_Channel\_\_: specifies the DAC channel

**Return value:**

- None:

**Description:**

- Disable the DAC channel.

**Parameters:**

- \_\_HANDLE\_\_: specifies the DAC handle
- \_\_DAC\_Channel\_\_: specifies the DAC channel.

**Return value:**

- None:

\_\_HAL\_DAC\_DISABLE

***DAC output buffer***

DAC\_OUTPUTBUFFER\_ENABLE

DAC\_OUTPUTBUFFER\_DISABLE

***DAC Private Macros***

IS\_DAC\_OUTPUT\_BUFFER\_STATE

IS\_DAC\_CHANNEL

IS\_DAC\_ALIGN

IS\_DAC\_DATA

DAC\_DHR12R1\_ALIGNMENT

DAC\_DHR12R2\_ALIGNMENT

DAC\_DHR12RD\_ALIGNMENT

## 12 HAL DAC Extension Driver

### 12.1 DACEx Firmware driver API description

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

#### 12.1.1 How to use this driver

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

#### 12.1.2 Extended features functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion.
- [HAL\\_DACEx\\_DualGetValue\(\)](#)
- [HAL\\_DACEx\\_TriangleWaveGenerate\(\)](#)
- [HAL\\_DACEx\\_NoiseWaveGenerate\(\)](#)
- [HAL\\_DACEx\\_DualSetValue\(\)](#)
- [HAL\\_DACEx\\_ConvCpltCallbackCh2\(\)](#)
- [HAL\\_DACEx\\_ConvHalfCpltCallbackCh2\(\)](#)
- [HAL\\_DACEx\\_ErrorCallbackCh2\(\)](#)

#### 12.1.3 HAL\_DACEx\_DualGetValue

Function Name	<b>uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)</b>
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b>: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• The selected DAC channel data output value.</li> </ul>

#### 12.1.4 HAL\_DACEx\_TriangleWaveGenerate

Function Name	<b>HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)</b>
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b>: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel</b>: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2</li> <li>• <b>Amplitude</b>: Select max triangle amplitude. This parameter can be one of the following values:  DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1  DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3  DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7  DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15  DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31  DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63  DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127  DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255  DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511  DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023  DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047  DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 12.1.5 HAL\_DACEx\_NoiseWaveGenerate

Function Name	<b>HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)</b>
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b>: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel</b>: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2</li> <li>• <b>Amplitude</b>: Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values:  DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation  DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation  DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation  DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR</li> </ul>

bit[3:0] for noise wave generation  
 DAC\_LFSRUNMASK\_BITS4\_0: Unmask DAC channel LFSR  
 bit[4:0] for noise wave generation  
 DAC\_LFSRUNMASK\_BITS5\_0: Unmask DAC channel LFSR  
 bit[5:0] for noise wave generation  
 DAC\_LFSRUNMASK\_BITS6\_0: Unmask DAC channel LFSR  
 bit[6:0] for noise wave generation  
 DAC\_LFSRUNMASK\_BITS7\_0: Unmask DAC channel LFSR  
 bit[7:0] for noise wave generation  
 DAC\_LFSRUNMASK\_BITS8\_0: Unmask DAC channel LFSR  
 bit[8:0] for noise wave generation  
 DAC\_LFSRUNMASK\_BITS9\_0: Unmask DAC channel LFSR  
 bit[9:0] for noise wave generation  
 DAC\_LFSRUNMASK\_BITS10\_0: Unmask DAC channel  
 LFSR bit[10:0] for noise wave generation  
 DAC\_LFSRUNMASK\_BITS11\_0: Unmask DAC channel  
 LFSR bit[11:0] for noise wave generation

Return values

- HAL status

### 12.1.6 HAL\_DACEx\_DualSetValue

**Function Name** `HAL_StatusTypeDef HAL_DACEx_DualSetValue  
(DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t  
Data1, uint32_t Data2)`

**Function Description** Set the specified data holding register value for dual DAC channel.

**Parameters**

- **hdac**: pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Alignment**: Specifies the data alignment for dual channel DAC. This parameter can be one of the following values:  
 DAC\_ALIGN\_8B\_R: 8bit right data alignment selected  
 DAC\_ALIGN\_12B\_L: 12bit left data alignment selected  
 DAC\_ALIGN\_12B\_R: 12bit right data alignment selected
- **Data1**: Data for DAC Channel2 to be loaded in the selected data holding register.
- **Data2**: Data for DAC Channel1 to be loaded in the selected data holding register.

**Return values**

- HAL status

**Notes**

- In dual mode, a unique register access is required to write in both DAC channels at the same time.

### 12.1.7 HAL\_DACEx\_ConvCpltCallbackCh2

**Function Name** `void HAL_DACEx_ConvCpltCallbackCh2  
(DAC_HandleTypeDef * hdac)`

**Function Description** Conversion complete callback in non blocking mode for Channel2.

**Parameters**

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

Return values • None

### 12.1.8 HAL\_DACEx\_ConvHalfCpltCallbackCh2

**Function Name** void HAL\_DACEx\_ConvHalfCpltCallbackCh2 (DAC\_HandleTypeDef \* hdac)

**Function Description** Conversion half DMA transfer callback in non blocking mode for Channel2.

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

**Return values** • None

### 12.1.9 HAL\_DACEx\_ErrorCallbackCh2

**Function Name** void HAL\_DACEx\_ErrorCallbackCh2 (DAC\_HandleTypeDef \* hdac)

**Function Description** Error DAC callback for Channel2.

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

**Return values** • None

## 12.2 DACEx Firmware driver defines

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

### 12.2.1 DACEx

DACEx

***DACEx Ifsrunmask triangleamplitude***

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

	generation
DAC_LFSRUNMASK_BITS8_0	Unmask DAC channel LFSR bit[8:0] for noise wave generation
DAC_LFSRUNMASK_BITS9_0	Unmask DAC channel LFSR bit[9:0] for noise wave generation
DAC_LFSRUNMASK_BITS10_0	Unmask DAC channel LFSR bit[10:0] for noise wave generation
DAC_LFSRUNMASK_BITS11_0	Unmask DAC channel LFSR bit[11:0] for noise wave generation
DAC_TRIANGLEAMPLITUDE_1	Select max triangle amplitude of 1
DAC_TRIANGLEAMPLITUDE_3	Select max triangle amplitude of 3
DAC_TRIANGLEAMPLITUDE_7	Select max triangle amplitude of 7
DAC_TRIANGLEAMPLITUDE_15	Select max triangle amplitude of 15
DAC_TRIANGLEAMPLITUDE_31	Select max triangle amplitude of 31
DAC_TRIANGLEAMPLITUDE_63	Select max triangle amplitude of 63
DAC_TRIANGLEAMPLITUDE_127	Select max triangle amplitude of 127
DAC_TRIANGLEAMPLITUDE_255	Select max triangle amplitude of 255
DAC_TRIANGLEAMPLITUDE_511	Select max triangle amplitude of 511
DAC_TRIANGLEAMPLITUDE_1023	Select max triangle amplitude of 1023
DAC_TRIANGLEAMPLITUDE_2047	Select max triangle amplitude of 2047
DAC_TRIANGLEAMPLITUDE_4095	Select max triangle amplitude of 4095

**DACEx Private Macros**

IS\_DAC\_TRIGGER

IS\_DAC\_GENERATE\_WAVE

IS\_DAC\_LFSR\_UNMASK\_TRIANGLE\_AMPLITUDE

IS\_DAC\_WAVE

**DAC trigger selection**

DAC_TRIGGER_NONE	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
DAC_TRIGGER_T6_TRGO	TIM6 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T7_TRGO	TIM7 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T5_TRGO	TIM5 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T2_TRGO	TIM2 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T4_TRGO	TIM4 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_EXT_IT9	EXTI Line9 event selected as external conversion trigger



for DAC channel

DAC\_TRIGGER\_SOFTWARE    Conversion started by software trigger for DAC channel

DAC\_TRIGGER\_T8\_TRGO    TIM8 TRGO selected as external conversion trigger for  
DAC channel

***DACEx wave generation***

DAC\_WAVEGENERATION\_NOISE

DAC\_WAVEGENERATION\_TRIANGLE

DAC\_WAVE\_NOISE

DAC\_WAVE\_TRIANGLE

## 13 HAL DMA Generic Driver

### 13.1 DMA Firmware driver registers structures

#### 13.1.1 DMA\_InitTypeDef

**DMA\_InitTypeDef** is defined in the stm32f1xx\_hal\_dma.h

##### Data Fields

- **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::Direction** Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of [DMA\\_Data\\_transfer\\_direction](#)
- **uint32\_t DMA\_InitTypeDef::PeriphInc** Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of [DMA\\_Peripheral\\_incremented\\_mode](#)
- **uint32\_t DMA\_InitTypeDef::MemInc** Specifies whether the memory address register should be incremented or not. This parameter can be a value of [DMA\\_Memory\\_incremented\\_mode](#)
- **uint32\_t DMA\_InitTypeDef::PeriphDataAlignment** Specifies the Peripheral data width. This parameter can be a value of [DMA\\_Peripheral\\_data\\_size](#)
- **uint32\_t DMA\_InitTypeDef::MemDataAlignment** Specifies the Memory data width. This parameter can be a value of [DMA\\_Memory\\_data\\_size](#)
- **uint32\_t DMA\_InitTypeDef::Mode** Specifies the operation mode of the DMAy Channelx. This parameter can be a value of [DMA\\_mode](#)  
**Note:** The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel
- **uint32\_t DMA\_InitTypeDef::Priority** Specifies the software priority for the DMAy Channelx. This parameter can be a value of [DMA\\_Priority\\_level](#)

#### 13.1.2 \_\_DMA\_HandleTypeDef

**\_\_DMA\_HandleTypeDef** is defined in the stm32f1xx\_hal\_dma.h

##### Data Fields

- **DMA\_Channel\_TypeDef \* Instance**
- **DMA\_InitTypeDef Init**
- **HAL\_LockTypeDef Lock**
- **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
- `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

## 13.2 DMA Firmware driver API description

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

### 13.2.1 How to use this driver

1. Enable and configure the peripheral to be connected to the DMA Channel (except for internal SRAM / FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and DMA requests .
2. For a given Channel, program the required configuration through the following parameters: Transfer Direction, Source and Destination data formats, Circular or Normal mode, Channel Priority level, Source and Destination Increment mode, using `HAL_DMA_Init()` function.
3. Use `HAL_DMA_GetState()` function to return the DMA state and `HAL_DMA_GetError()` in case of error detection.
4. Use `HAL_DMA_Abort()` function to abort the current transfer In Memory-to-Memory transfer mode, Circular mode is not allowed.

#### Polling mode IO operation

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

#### Interrupt mode IO operation

- Configure the DMA interrupt priority using `HAL_NVIC_SetPriority()`
- Enable the DMA IRQ handler using `HAL_NVIC_EnableIRQ()`

- Use HAL\_DMA\_Start\_IT() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use HAL\_DMAy\_Channelx\_IRQHandler() called under DMA\_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL\_DMA\_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback and XferErrorCallback (i.e a member of DMA handle structure).

### DMA HAL driver macros list

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

- \_\_HAL\_DMA\_ENABLE: Enable the specified DMA Channel.
- \_\_HAL\_DMA\_DISABLE: Disable the specified DMA Channel.
- \_\_HAL\_DMA\_GET\_FLAG: Get the DMA Channel pending flags.
- \_\_HAL\_DMA\_CLEAR\_FLAG: Clear the DMA Channel pending flags.
- \_\_HAL\_DMA\_ENABLE\_IT: Enable the specified DMA Channel interrupts.
- \_\_HAL\_DMA\_DISABLE\_IT: Disable the specified DMA Channel interrupts.
- \_\_HAL\_DMA\_GET\_IT\_SOURCE: Check whether the specified DMA Channel interrupt has occurred or not.



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

## 13.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize the DMA Channel source and destination addresses, incrementation and data sizes, transfer direction, circular/normal mode selection, memory-to-memory mode selection and Channel priority value.

The HAL\_DMA\_Init() function follows the DMA configuration procedures as described in reference manual.

- [HAL\\_DMA\\_Init\(\)](#)
- [HAL\\_DMA\\_DeInit\(\)](#)

## 13.2.3 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt
- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request
- [HAL\\_DMA\\_Start\(\)](#)
- [HAL\\_DMA\\_Start\\_IT\(\)](#)
- [HAL\\_DMA\\_Abort\(\)](#)
- [HAL\\_DMA\\_PollForTransfer\(\)](#)
- [HAL\\_DMA\\_IRQHandler\(\)](#)

## 13.2.4 State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code
- [HAL\\_DMA\\_GetState\(\)](#)
- [HAL\\_DMA\\_GetError\(\)](#)

## 13.2.5 HAL\_DMA\_Init

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)</b>
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"> <li>• <b>hdma</b>: Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 13.2.6 HAL\_DMA\_DeInit

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

## 13.2.7 HAL\_DMA\_Start

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)</b>
Function Description	Starts the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b>: : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.</li> <li>• <b>SrcAddress</b>: The source memory Buffer address</li> <li>• <b>DstAddress</b>: The destination memory Buffer address</li> <li>• <b>DataLength</b>: The length of data to be transferred from source to destination</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 13.2.8 HAL\_DMA\_Start\_IT

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

### 13.2.9 HAL\_DMA\_Abort

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Abort</b> (DMA_HandleTypeDef * hdma)
Function Description	Aborts the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b>: : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• After disabling a DMA Channel, a check for wait until the DMA Channel is effectively disabled is added. If a Channel is disabled while a data transfer is ongoing, the current data will be transferred and the Channel will be effectively disabled only after the transfer of this single data is finished.</li> </ul>

### 13.2.10 HAL\_DMA\_PollForTransfer

Function Name	<b>HAL_StatusTypeDef HAL_DMA_PollForTransfer</b> (DMA_HandleTypeDef * hdma, uint32_t CompleteLevel, uint32_t Timeout)
Function Description	Polling for transfer complete.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b>: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.</li> <li>• <b>CompleteLevel</b>: Specifies the DMA level complete.</li> <li>• <b>Timeout</b>: Timeout duration.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**13.2.11 HAL\_DMA\_IRQHandler**

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

**13.2.12 HAL\_DMA\_GetState**

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

**13.2.13 HAL\_DMA\_GetError**

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

**13.3 DMA Firmware driver defines**

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

**13.3.1 DMA**

DMA

***DMA Data buffer size***

IS\_DMA\_BUFFER\_SIZE

***DMA Data transfer direction***

DMA\_PERIPH\_TO\_MEMORY    Peripheral to memory direction

DMA\_MEMORY\_TO\_PERIPH    Memory to peripheral direction

DMA\_MEMORY\_TO\_MEMORY    Memory to memory direction

IS\_DMA\_DIRECTION

**DMA Error Codes**

HAL_DMA_ERROR_NONE	No error
HAL_DMA_ERROR_TE	Transfer error
HAL_DMA_ERROR_TIMEOUT	Timeout error

**DMA Exported Macros**

`__HAL_DMA_RESET_HANDLE_STATE` **Description:**

- Reset DMA handle state.

**Parameters:**

- `__HANDLE__`: DMA handle.

**Return value:**

- None:

`__HAL_DMA_ENABLE`

**Description:**

- Enable the specified DMA Channel.

**Parameters:**

- `__HANDLE__`: DMA handle

**Return value:**

- None.:

`__HAL_DMA_DISABLE`

**Description:**

- Disable the specified DMA Channel.

**Parameters:**

- `__HANDLE__`: DMA handle

**Return value:**

- None.:

`__HAL_DMA_ENABLE_IT`

**Description:**

- Enables the specified DMA Channel interrupts.

**Parameters:**

- `__HANDLE__`: DMA handle
- `__INTERRUPT__`: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - `DMA_IT_TC`: Transfer complete interrupt mask
  - `DMA_IT_HT`: Half transfer complete interrupt mask
  - `DMA_IT_TE`: Transfer error interrupt mask

**Return value:**



`__HAL_DMA_DISABLE_IT`

- None:

**Description:**

- Disables the specified DMA Channel interrupts.

**Parameters:**

- `__HANDLE__`: DMA handle
- `__INTERRUPT__`: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - `DMA_IT_TC`: Transfer complete interrupt mask
  - `DMA_IT_HT`: Half transfer complete interrupt mask
  - `DMA_IT_TE`: Transfer error interrupt mask

**Return value:**

- None:

`__HAL_DMA_GET_IT_SOURCE`

**Description:**

- Checks whether the specified DMA Channel interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: DMA handle
- `__INTERRUPT__`: specifies the DMA interrupt source to check. This parameter can be one of the following values:
  - `DMA_IT_TC`: Transfer complete interrupt mask
  - `DMA_IT_HT`: Half transfer complete interrupt mask
  - `DMA_IT_TE`: Transfer error interrupt mask

**Return value:**

- The: state of DMA\_IT (SET or RESET).

***DMA flag definitions***

`DMA_FLAG_GL1`

`DMA_FLAG_TC1`

`DMA_FLAG_HT1`

`DMA_FLAG_TE1`

`DMA_FLAG_GL2`

`DMA_FLAG_TC2`

`DMA_FLAG_HT2`

`DMA_FLAG_TE2`

DMA\_FLAG\_GL3

DMA\_FLAG\_TC3

DMA\_FLAG\_HT3

DMA\_FLAG\_TE3

DMA\_FLAG\_GL4

DMA\_FLAG\_TC4

DMA\_FLAG\_HT4

DMA\_FLAG\_TE4

DMA\_FLAG\_GL5

DMA\_FLAG\_TC5

DMA\_FLAG\_HT5

DMA\_FLAG\_TE5

DMA\_FLAG\_GL6

DMA\_FLAG\_TC6

DMA\_FLAG\_HT6

DMA\_FLAG\_TE6

DMA\_FLAG\_GL7

DMA\_FLAG\_TC7

DMA\_FLAG\_HT7

DMA\_FLAG\_TE7

***DMA interrupt enable definitions***

DMA\_IT\_TC

DMA\_IT\_HT

DMA\_IT\_TE

***DMA Memory data size***

DMA\_MDATAALIGN\_BYTE      Memory data alignment : Byte

DMA\_MDATAALIGN\_HALFWORD      Memory data alignment : HalfWord

DMA\_MDATAALIGN\_WORD      Memory data alignment : Word

IS\_DMA\_MEMORY\_DATA\_SIZE

***DMA Memory incremented mode***

DMA\_MINC\_ENABLE      Memory increment mode Enable

DMA\_MINC\_DISABLE      Memory increment mode Disable

IS\_DMA\_MEMORY\_INC\_STATE

***DMA mode***

DMA\_NORMAL      Normal Mode

DMA\_CIRCULAR      Circular Mode

IS\_DMA\_MODE

**DMA Peripheral data size**

DMA\_PDATAALIGN\_BYTE                      Peripheral data alignment : Byte

DMA\_PDATAALIGN\_HALFWORD                  Peripheral data alignment : HalfWord

DMA\_PDATAALIGN\_WORD                      Peripheral data alignment : Word

IS\_DMA\_PERIPHERAL\_DATA\_SIZE

**DMA Peripheral incremented mode**

DMA\_PINC\_ENABLE                            Peripheral increment mode Enable

DMA\_PINC\_DISABLE                           Peripheral increment mode Disable

IS\_DMA\_PERIPHERAL\_INC\_STATE

**DMA Priority level**

DMA\_PRIORITY\_LOW                          Priority level : Low

DMA\_PRIORITY\_MEDIUM                      Priority level : Medium

DMA\_PRIORITY\_HIGH                         Priority level : High

DMA\_PRIORITY\_VERY\_HIGH                  Priority level : Very\_High

IS\_DMA\_PRIORITY

**DMA Private Constants**

HAL\_TIMEOUT\_DMA\_ABORT

## 14 HAL DMA Extension Driver

### 14.1 DMAEx Firmware driver defines

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

#### 14.1.1 DMAEx

DMAEx

*DMAEx High density and XL density product devices*

`__HAL_DMA_GET_TC_FLAG_INDEX`

`__HAL_DMA_GET_HT_FLAG_INDEX`

**Description:**

- Returns the current DMA Channel half transfer complete flag.

**Parameters:**

- `__HANDLE__`: DMA handle

**Return value:**

- The: specified half transfer complete flag index.

`__HAL_DMA_GET_TE_FLAG_INDEX`

**Description:**

- Returns the current DMA Channel transfer error flag.

**Parameters:**

- `__HANDLE__`: DMA handle

**Return value:**

- The: specified transfer error flag index.

`__HAL_DMA_GET_FLAG`

**Description:**

- Get the DMA Channel pending flags.

**Parameters:**

- `__HANDLE__`: DMA handle
- `__FLAG__`: Get the specified flag. This parameter can be any combination of the following values:
  - `DMA_FLAG_TCx`: Transfer complete flag
  - `DMA_FLAG_HTx`: Half transfer complete flag
  - `DMA_FLAG_TEx`: Transfer error flag
 Where x can be 1\_7 or 1\_5 (depending on DMA1 or DMA2) to select the DMA Channel flag.

**Return value:**

---

`__HAL_DMA_CLEAR_FLAG`

- The: state of FLAG (SET or RESET).

**Description:**

- Clears the DMA Channel pending flags.

**Parameters:**

- `__HANDLE__`: DMA handle
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `DMA_FLAG_TCx`: Transfer complete flag
  - `DMA_FLAG_HTx`: Half transfer complete flag
  - `DMA_FLAG_TEx`: Transfer error flagWhere x can be 1\_7 or 1\_5 (depending on DMA1 or DMA2) to select the DMA Channel flag.

**Return value:**

- None:

## 15 HAL ETH Generic Driver

### 15.1 ETH Firmware driver registers structures

#### 15.1.1 ETH\_InitTypeDef

*ETH\_InitTypeDef* is defined in the stm32f1xx\_hal\_eth.h

##### Data Fields

- *uint32\_t* **AutoNegotiation**
- *uint32\_t* **Speed**
- *uint32\_t* **DuplexMode**
- *uint16\_t* **PhyAddress**
- *uint8\_t* \* **MACAddr**
- *uint32\_t* **RxMode**
- *uint32\_t* **ChecksumMode**
- *uint32\_t* **MedialInterface**

##### Field Documentation

- *uint32\_t* **ETH\_InitTypeDef::AutoNegotiation** Selects or not the AutoNegotiation mode for the external PHY. The AutoNegotiation allows an automatic setting of the Speed (10/100Mbps) and the mode (half/full-duplex). This parameter can be a value of [ETH\\_AutoNegotiation](#)
- *uint32\_t* **ETH\_InitTypeDef::Speed** Sets the Ethernet speed: 10/100 Mbps. This parameter can be a value of [ETH\\_Speed](#)
- *uint32\_t* **ETH\_InitTypeDef::DuplexMode** Selects the MAC duplex mode: Half-Duplex or Full-Duplex mode. This parameter can be a value of [ETH\\_Duplex\\_Mode](#)
- *uint16\_t* **ETH\_InitTypeDef::PhyAddress** Ethernet PHY address. This parameter must be a number between Min\_Data = 0 and Max\_Data = 32
- *uint8\_t*\* **ETH\_InitTypeDef::MACAddr** MAC Address of used Hardware: must be pointer on an array of 6 bytes
- *uint32\_t* **ETH\_InitTypeDef::RxMode** Selects the Ethernet Rx mode: Polling mode, Interrupt mode. This parameter can be a value of [ETH\\_Rx\\_Mode](#)
- *uint32\_t* **ETH\_InitTypeDef::ChecksumMode** Selects if the checksum is checked by hardware or by software. This parameter can be a value of [ETH\\_Checksum\\_Mode](#)
- *uint32\_t* **ETH\_InitTypeDef::MedialInterface** Selects the media-independent interface or the reduced media-independent interface. This parameter can be a value of [ETH\\_Media\\_Interface](#)

#### 15.1.2 ETH\_MACInitTypeDef

*ETH\_MACInitTypeDef* is defined in the stm32f1xx\_hal\_eth.h

##### Data Fields

- *uint32\_t* **Watchdog**
- *uint32\_t* **Jabber**
- *uint32\_t* **InterFrameGap**
- *uint32\_t* **CarrierSense**
- *uint32\_t* **ReceiveOwn**

- `uint32_t LoopbackMode`
- `uint32_t ChecksumOffload`
- `uint32_t RetryTransmission`
- `uint32_t AutomaticPadCRCStrip`
- `uint32_t BackOffLimit`
- `uint32_t DeferralCheck`
- `uint32_t ReceiveAll`
- `uint32_t SourceAddrFilter`
- `uint32_t PassControlFrames`
- `uint32_t BroadcastFramesReception`
- `uint32_t DestinationAddrFilter`
- `uint32_t PromiscuousMode`
- `uint32_t MulticastFramesFilter`
- `uint32_t UnicastFramesFilter`
- `uint32_t HashTableHigh`
- `uint32_t HashTableLow`
- `uint32_t PauseTime`
- `uint32_t ZeroQuantaPause`
- `uint32_t PauseLowThreshold`
- `uint32_t UnicastPauseFrameDetect`
- `uint32_t ReceiveFlowControl`
- `uint32_t TransmitFlowControl`
- `uint32_t VLANTagComparison`
- `uint32_t VLANTagIdentifier`

#### Field Documentation

- `uint32_t ETH_MACInitTypeDef::Watchdog` Selects or not the Watchdog timer. When enabled, the MAC allows no more than 2048 bytes to be received. When disabled, the MAC can receive up to 16384 bytes. This parameter can be a value of [ETH\\_Watchdog](#)
- `uint32_t ETH_MACInitTypeDef::Jabber` Selects or not Jabber timer. When enabled, the MAC allows no more than 2048 bytes to be sent. When disabled, the MAC can send up to 16384 bytes. This parameter can be a value of [ETH\\_Jabber](#)
- `uint32_t ETH_MACInitTypeDef::InterFrameGap` Selects the minimum IFG between frames during transmission. This parameter can be a value of [ETH\\_Inter\\_Frame\\_Gap](#)
- `uint32_t ETH_MACInitTypeDef::CarrierSense` Selects or not the Carrier Sense. This parameter can be a value of [ETH\\_Carrier\\_Sense](#)
- `uint32_t ETH_MACInitTypeDef::ReceiveOwn` Selects or not the ReceiveOwn. ReceiveOwn allows the reception of frames when the TX\_EN signal is asserted in Half-Duplex mode. This parameter can be a value of [ETH\\_Receive\\_Own](#)
- `uint32_t ETH_MACInitTypeDef::LoopbackMode` Selects or not the internal MAC MII Loopback mode. This parameter can be a value of [ETH\\_Loop\\_Back\\_Mode](#)
- `uint32_t ETH_MACInitTypeDef::ChecksumOffload` Selects or not the IPv4 checksum checking for received frame payloads' TCP/UDP/ICMP headers. This parameter can be a value of [ETH\\_Checksum\\_Offload](#)
- `uint32_t ETH_MACInitTypeDef::RetryTransmission` Selects or not the MAC attempt retries transmission, based on the settings of BL, when a collision occurs (Half-Duplex mode). This parameter can be a value of [ETH\\_Retry\\_Transmission](#)
- `uint32_t ETH_MACInitTypeDef::AutomaticPadCRCStrip` Selects or not the Automatic MAC Pad/CRC Stripping. This parameter can be a value of [ETH\\_Automatic\\_Pad\\_CRC\\_Strip](#)

- **uint32\_t ETH\_MACInitTypeDef::BackOffLimit** Selects the BackOff limit value. This parameter can be a value of [ETH\\_Back\\_Off\\_Limit](#)
- **uint32\_t ETH\_MACInitTypeDef::DeferralCheck** Selects or not the deferral check function (Half-Duplex mode). This parameter can be a value of [ETH\\_Deferral\\_Check](#)
- **uint32\_t ETH\_MACInitTypeDef::ReceiveAll** Selects or not all frames reception by the MAC (No filtering). This parameter can be a value of [ETH\\_Receive\\_All](#)
- **uint32\_t ETH\_MACInitTypeDef::SourceAddrFilter** Selects the Source Address Filter mode. This parameter can be a value of [ETH\\_Source\\_Addr\\_Filter](#)
- **uint32\_t ETH\_MACInitTypeDef::PassControlFrames** Sets the forwarding mode of the control frames (including unicast and multicast PAUSE frames) This parameter can be a value of [ETH\\_Pass\\_Control\\_Frames](#)
- **uint32\_t ETH\_MACInitTypeDef::BroadcastFramesReception** Selects or not the reception of Broadcast Frames. This parameter can be a value of [ETH\\_Broadcast\\_Frames\\_Reception](#)
- **uint32\_t ETH\_MACInitTypeDef::DestinationAddrFilter** Sets the destination filter mode for both unicast and multicast frames. This parameter can be a value of [ETH\\_Destination\\_Addr\\_Filter](#)
- **uint32\_t ETH\_MACInitTypeDef::PromiscuousMode** Selects or not the Promiscuous Mode This parameter can be a value of [ETH\\_Promiscuous\\_Mode](#)
- **uint32\_t ETH\_MACInitTypeDef::MulticastFramesFilter** Selects the Multicast Frames filter mode: None/HashTableFilter/PerfectFilter/PerfectHashTableFilter. This parameter can be a value of [ETH\\_Multicast\\_Frames\\_Filter](#)
- **uint32\_t ETH\_MACInitTypeDef::UnicastFramesFilter** Selects the Unicast Frames filter mode: HashTableFilter/PerfectFilter/PerfectHashTableFilter. This parameter can be a value of [ETH\\_Unicast\\_Frames\\_Filter](#)
- **uint32\_t ETH\_MACInitTypeDef::HashTableHigh** This field holds the higher 32 bits of Hash table. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFFFFFF
- **uint32\_t ETH\_MACInitTypeDef::HashTableLow** This field holds the lower 32 bits of Hash table. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFFFFFF
- **uint32\_t ETH\_MACInitTypeDef::PauseTime** This field holds the value to be used in the Pause Time field in the transmit control frame. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFF
- **uint32\_t ETH\_MACInitTypeDef::ZeroQuantaPause** Selects or not the automatic generation of Zero-Quanta Pause Control frames. This parameter can be a value of [ETH\\_Zero\\_Quanta\\_Pause](#)
- **uint32\_t ETH\_MACInitTypeDef::PauseLowThreshold** This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Frame. This parameter can be a value of [ETH\\_Pause\\_Low\\_Threshold](#)
- **uint32\_t ETH\_MACInitTypeDef::UnicastPauseFrameDetect** Selects or not the MAC detection of the Pause frames (with MAC Address0 unicast address and unique multicast address). This parameter can be a value of [ETH\\_Unicast\\_Pause\\_Frame\\_Detect](#)
- **uint32\_t ETH\_MACInitTypeDef::ReceiveFlowControl** Enables or disables the MAC to decode the received Pause frame and disable its transmitter for a specified time (Pause Time) This parameter can be a value of [ETH\\_Receive\\_Flow\\_Control](#)
- **uint32\_t ETH\_MACInitTypeDef::TransmitFlowControl** Enables or disables the MAC to transmit Pause frames (Full-Duplex mode) or the MAC back-pressure operation (Half-Duplex mode) This parameter can be a value of [ETH\\_Transmit\\_Flow\\_Control](#)
- **uint32\_t ETH\_MACInitTypeDef::VLANTagComparison** Selects the 12-bit VLAN identifier or the complete 16-bit VLAN tag for comparison and filtering. This parameter can be a value of [ETH\\_VLAN\\_Tag\\_Comparison](#)



- **`uint32_t ETH_MACInitTypeDef::VLANTagIdentifier`** Holds the VLAN tag identifier for receive frames

### 15.1.3 ETH\_DMAInitTypeDef

**`ETH_DMAInitTypeDef`** is defined in the `stm32f1xx_hal_eth.h`

#### Data Fields

- **`uint32_t DropTCPIPChecksumErrorFrame`**
- **`uint32_t ReceiveStoreForward`**
- **`uint32_t FlushReceivedFrame`**
- **`uint32_t TransmitStoreForward`**
- **`uint32_t TransmitThresholdControl`**
- **`uint32_t ForwardErrorFrames`**
- **`uint32_t ForwardUndersizedGoodFrames`**
- **`uint32_t ReceiveThresholdControl`**
- **`uint32_t SecondFrameOperate`**
- **`uint32_t AddressAlignedBeats`**
- **`uint32_t FixedBurst`**
- **`uint32_t RxDMABurstLength`**
- **`uint32_t TxDMABurstLength`**
- **`uint32_t DescriptorSkipLength`**
- **`uint32_t DMAArbitration`**

#### Field Documentation

- **`uint32_t ETH_DMAInitTypeDef::DropTCPIPChecksumErrorFrame`** Selects or not the Dropping of TCP/IP Checksum Error Frames. This parameter can be a value of [\*\*`ETH\_Drop\_TCP\_IP\_Checksum\_Error\_Frame`\*\*](#)
- **`uint32_t ETH_DMAInitTypeDef::ReceiveStoreForward`** Enables or disables the Receive store and forward mode. This parameter can be a value of [\*\*`ETH\_Receive\_Store\_Forward`\*\*](#)
- **`uint32_t ETH_DMAInitTypeDef::FlushReceivedFrame`** Enables or disables the flushing of received frames. This parameter can be a value of [\*\*`ETH\_Flush\_Received\_Frame`\*\*](#)
- **`uint32_t ETH_DMAInitTypeDef::TransmitStoreForward`** Enables or disables Transmit store and forward mode. This parameter can be a value of [\*\*`ETH\_Transmit\_Store\_Forward`\*\*](#)
- **`uint32_t ETH_DMAInitTypeDef::TransmitThresholdControl`** Selects or not the Transmit Threshold Control. This parameter can be a value of [\*\*`ETH\_Transmit\_Threshold\_Control`\*\*](#)
- **`uint32_t ETH_DMAInitTypeDef::ForwardErrorFrames`** Selects or not the forward to the DMA of erroneous frames. This parameter can be a value of [\*\*`ETH\_Forward\_Error\_Frames`\*\*](#)
- **`uint32_t ETH_DMAInitTypeDef::ForwardUndersizedGoodFrames`** Enables or disables the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC) This parameter can be a value of [\*\*`ETH\_Forward\_Undersized\_Good\_Frames`\*\*](#)
- **`uint32_t ETH_DMAInitTypeDef::ReceiveThresholdControl`** Selects the threshold level of the Receive FIFO. This parameter can be a value of [\*\*`ETH\_Receive\_Threshold\_Control`\*\*](#)

- ***uint32\_t ETH\_DMAInitTypeDef::SecondFrameOperate*** Selects or not the Operate on second frame mode, which allows the DMA to process a second frame of Transmit data even before obtaining the status for the first frame. This parameter can be a value of [ETH\\_Second\\_Frame\\_Operate](#)
- ***uint32\_t ETH\_DMAInitTypeDef::AddressAlignedBeats*** Enables or disables the Address Aligned Beats. This parameter can be a value of [ETH\\_Address\\_Aligned\\_Beats](#)
- ***uint32\_t ETH\_DMAInitTypeDef::FixedBurst*** Enables or disables the AHB Master interface fixed burst transfers. This parameter can be a value of [ETH\\_Fixed\\_Burst](#)
- ***uint32\_t ETH\_DMAInitTypeDef::RxDMA BurstLength*** Indicates the maximum number of beats to be transferred in one Rx DMA transaction. This parameter can be a value of [ETH\\_Rx\\_DMA\\_Burst\\_Length](#)
- ***uint32\_t ETH\_DMAInitTypeDef::TxDMA BurstLength*** Indicates the maximum number of beats to be transferred in one Tx DMA transaction. This parameter can be a value of [ETH\\_Tx\\_DMA\\_Burst\\_Length](#)
- ***uint32\_t ETH\_DMAInitTypeDef::DescriptorSkipLength*** Specifies the number of word to skip between two unchained descriptors (Ring mode) This parameter must be a number between Min\_Data = 0 and Max\_Data = 32
- ***uint32\_t ETH\_DMAInitTypeDef::DMAArbitration*** Selects the DMA Tx/Rx arbitration. This parameter can be a value of [ETH\\_DMA\\_Arbitration](#)

#### 15.1.4 ETH\_DMADescTypeDef

*ETH\_DMADescTypeDef* is defined in the stm32f1xx\_hal\_eth.h

##### Data Fields

- ***\_\_IO uint32\_t Status***
- ***uint32\_t ControlBufferSize***
- ***uint32\_t Buffer1Addr***
- ***uint32\_t Buffer2NextDescAddr***

##### Field Documentation

- ***\_\_IO uint32\_t ETH\_DMADescTypeDef::Status*** Status
- ***uint32\_t ETH\_DMADescTypeDef::ControlBufferSize*** Control and Buffer1, Buffer2 lengths
- ***uint32\_t ETH\_DMADescTypeDef::Buffer1Addr*** Buffer1 address pointer
- ***uint32\_t ETH\_DMADescTypeDef::Buffer2NextDescAddr*** Buffer2 or next descriptor address pointer

#### 15.1.5 ETH\_DMARxFramInfos

*ETH\_DMARxFramInfos* is defined in the stm32f1xx\_hal\_eth.h

##### Data Fields

- ***ETH\_DMADescTypeDef \* FSRxDesc***
- ***ETH\_DMADescTypeDef \* LSRxDesc***
- ***uint32\_t SegCount***
- ***uint32\_t length***
- ***uint32\_t buffer***

**Field Documentation**

- ***ETH\_DMADescTypeDef\* ETH\_DMARxFramelInfos::FSRxDesc*** First Segment Rx Desc
- ***ETH\_DMADescTypeDef\* ETH\_DMARxFramelInfos::LSRxDesc*** Last Segment Rx Desc
- ***uint32\_t ETH\_DMARxFramelInfos::SegCount*** Segment count
- ***uint32\_t ETH\_DMARxFramelInfos::length*** Frame length
- ***uint32\_t ETH\_DMARxFramelInfos::buffer*** Frame buffer

**15.1.6 ETH\_HandleTypeDef**

***ETH\_HandleTypeDef*** is defined in the stm32f1xx\_hal\_eth.h

**Data Fields**

- ***ETH\_TypeDef \* Instance***
- ***ETH\_InitTypeDef Init***
- ***uint32\_t LinkStatus***
- ***ETH\_DMADescTypeDef \* RxDesc***
- ***ETH\_DMADescTypeDef \* TxDesc***
- ***ETH\_DMARxFramelInfos RxFrameInfos***
- ***\_\_IO HAL\_ETH\_StateTypeDef State***
- ***HAL\_LockTypeDef Lock***

**Field Documentation**

- ***ETH\_TypeDef\* ETH\_HandleTypeDef::Instance*** Register base address
- ***ETH\_InitTypeDef ETH\_HandleTypeDef::Init*** Ethernet Init Configuration
- ***uint32\_t ETH\_HandleTypeDef::LinkStatus*** Ethernet link status
- ***ETH\_DMADescTypeDef\* ETH\_HandleTypeDef::RxDesc*** Rx descriptor to Get
- ***ETH\_DMADescTypeDef\* ETH\_HandleTypeDef::TxDesc*** Tx descriptor to Set
- ***ETH\_DMARxFramelInfos ETH\_HandleTypeDef::RxFrameInfos*** last Rx frame infos
- ***\_\_IO HAL\_ETH\_StateTypeDef ETH\_HandleTypeDef::State*** ETH communication state
- ***HAL\_LockTypeDef ETH\_HandleTypeDef::Lock*** ETH Lock

**15.2 ETH Firmware driver API description**

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

**15.2.1 How to use this driver**

1. Declare a ***ETH\_HandleTypeDef*** handle structure, for example: ***ETH\_HandleTypeDef heth;***
2. Fill parameters of ***Init*** structure in ***heth*** handle
3. Call ***HAL\_ETH\_Init()*** API to initialize the Ethernet peripheral (MAC, DMA, ...)
4. Initialize the ETH low level resources through the ***HAL\_ETH\_Mspinit()*** API:
  - a. Enable the Ethernet interface clock using
    - ***\_\_HAL\_RCC\_ETHMAC\_CLK\_ENABLE();***

- `__HAL_RCC_ETHMACTX_CLK_ENABLE();`
- `__HAL_RCC_ETHMACRX_CLK_ENABLE();`
- b. Initialize the related GPIO clocks
- c. Configure Ethernet pin-out
- d. Configure Ethernet NVIC interrupt (IT mode)
- 5. Initialize Ethernet DMA Descriptors in chain mode and point to allocated buffers:
  - a. `HAL_ETH_DMATxDescListInit();` for Transmission process
  - b. `HAL_ETH_DMARxDescListInit();` for Reception process
- 6. Enable MAC and DMA transmission and reception:
  - a. `HAL_ETH_Start();`
- 7. Prepare ETH DMA TX Descriptors and give the hand to ETH DMA to transfer the frame to MAC TX FIFO:
  - a. `HAL_ETH_TransmitFrame();`
- 8. Poll for a received frame in ETH RX DMA Descriptors and get received frame parameters
  - a. `HAL_ETH_GetReceivedFrame();` (should be called into an infinite loop)
- 9. Get a received frame when an ETH RX interrupt occurs:
  - a. `HAL_ETH_GetReceivedFrame_IT();` (called in IT mode only)
- 10. Communicate with external PHY device:
  - a. Read a specific register from the PHY `HAL_ETH_ReadPHYRegister();`
  - b. Write data to a specific RHY register: `HAL_ETH_WritePHYRegister();`
- 11. Configure the Ethernet MAC after ETH peripheral initialization  
`HAL_ETH_ConfigMAC();` all MAC parameters should be filled.
- 12. Configure the Ethernet DMA after ETH peripheral initialization  
`HAL_ETH_ConfigDMA();` all DMA parameters should be filled. The PTP protocol and the DMA descriptors ring mode are not supported in this driver

### 15.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the Ethernet peripheral
- De-initialize the Ethernet peripheral
- [`HAL\_ETH\_Init\(\)`](#)
- [`HAL\_ETH\_DeInit\(\)`](#)
- [`HAL\_ETH\_DMATxDescListInit\(\)`](#)
- [`HAL\_ETH\_DMARxDescListInit\(\)`](#)
- [`HAL\_ETH\_MspInit\(\)`](#)
- [`HAL\_ETH\_MspDeInit\(\)`](#)

### 15.2.3 IO operation functions

This section provides functions allowing to:

- Transmit a frame `HAL_ETH_TransmitFrame();`
- Receive a frame `HAL_ETH_GetReceivedFrame();`  
`HAL_ETH_GetReceivedFrame_IT();`
- Read from an External PHY register `HAL_ETH_ReadPHYRegister();`
- Write to an External PHY register `HAL_ETH_WritePHYRegister();`
- [`HAL\_ETH\_TransmitFrame\(\)`](#)
- [`HAL\_ETH\_GetReceivedFrame\(\)`](#)
- [`HAL\_ETH\_GetReceivedFrame\_IT\(\)`](#)
- [`HAL\_ETH\_IRQHandler\(\)`](#)

- [HAL\\_ETH\\_TxCpltCallback\(\)](#)
- [HAL\\_ETH\\_RxCpltCallback\(\)](#)
- [HAL\\_ETH\\_ErrorCallback\(\)](#)
- [HAL\\_ETH\\_ReadPHYRegister\(\)](#)
- [HAL\\_ETH\\_WritePHYRegister\(\)](#)

## 15.2.4 Peripheral Control functions

This section provides functions allowing to:

- Enable MAC and DMA transmission and reception. `HAL_ETH_Start()`;
- Disable MAC and DMA transmission and reception. `HAL_ETH_Stop()`;
- Set the MAC configuration in runtime mode `HAL_ETH_ConfigMAC()`;
- Set the DMA configuration in runtime mode `HAL_ETH_ConfigDMA()`;
- [HAL\\_ETH\\_Start\(\)](#)
- [HAL\\_ETH\\_Stop\(\)](#)
- [HAL\\_ETH\\_ConfigMAC\(\)](#)
- [HAL\\_ETH\\_ConfigDMA\(\)](#)

## 15.2.5 Peripheral State functions

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

- Get the ETH handle state: `HAL_ETH_GetState()`;
- [HAL\\_ETH\\_GetState\(\)](#)

## 15.2.6 HAL\_ETH\_Init

Function Name	<b>HAL_StatusTypeDef HAL_ETH_Init (ETH_HandleTypeDef * heth)</b>
Function Description	Initializes the Ethernet MAC and DMA according to default parameters.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 15.2.7 HAL\_ETH\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_ETH_DeInit (ETH_HandleTypeDef * heth)</b>
Function Description	De-Initializes the ETH peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**15.2.8 HAL\_ETH\_DMATxDescListInit**

Function Name	<b>HAL_StatusTypeDef HAL_ETH_DMATxDescListInit (ETH_HandleTypeDef * heth, ETH_DMADescTypeDef * DMATxDescTab, uint8_t * TxBuff, uint32_t TxBuffCount)</b>
Function Description	Initializes the DMA Tx descriptors in chain mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li>• <b>DMATxDescTab</b>: Pointer to the first Tx desc list</li> <li>• <b>TxBuff</b>: Pointer to the first TxBuffer list</li> <li>• <b>TxBuffCount</b>: Number of the used Tx desc in the list</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**15.2.9 HAL\_ETH\_DMARxDescListInit**

Function Name	<b>HAL_StatusTypeDef HAL_ETH_DMARxDescListInit (ETH_HandleTypeDef * heth, ETH_DMADescTypeDef * DMARxDescTab, uint8_t * RxBuff, uint32_t RxBuffCount)</b>
Function Description	Initializes the DMA Rx descriptors in chain mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li>• <b>DMARxDescTab</b>: Pointer to the first Rx desc list</li> <li>• <b>RxBuff</b>: Pointer to the first RxBuffer list</li> <li>• <b>RxBuffCount</b>: Number of the used Rx desc in the list</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**15.2.10 HAL\_ETH\_MspInit**

Function Name	<b>void HAL_ETH_MspInit (ETH_HandleTypeDef * heth)</b>
Function Description	Initializes the ETH MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**15.2.11 HAL\_ETH\_MspDeInit**

Function Name	<b>void HAL_ETH_MspDeInit (ETH_HandleTypeDef * heth)</b>
Function Description	DeInitializes ETH MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**15.2.12 HAL\_ETH\_TransmitFrame**

Function Name	<b>HAL_StatusTypeDef HAL_ETH_TransmitFrame (ETH_HandleTypeDef * heth, uint32_t FrameLength)</b>
Function Description	Sends an Ethernet frame.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li><b>FrameLength</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**15.2.13 HAL\_ETH\_GetReceivedFrame**

Function Name	<b>HAL_StatusTypeDef HAL_ETH_GetReceivedFrame (ETH_HandleTypeDef * heth)</b>
Function Description	Checks for received frames.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**15.2.14 HAL\_ETH\_GetReceivedFrame\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_ETH_GetReceivedFrame_IT (ETH_HandleTypeDef * heth)</b>
Function Description	Gets the Received frame in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**15.2.15 HAL\_ETH\_IRQHandler**

Function Name	<b>void HAL_ETH_IRQHandler (ETH_HandleTypeDef * heth)</b>
Function Description	This function handles ETH interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**15.2.16 HAL\_ETH\_TxCpltCallback**

Function Name	<b>void HAL_ETH_TxCpltCallback (ETH_HandleTypeDef * heth)</b>
Function Description	Tx Transfer completed callbacks.

Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 15.2.17 HAL\_ETH\_RxCpltCallback

Function Name	<b>void HAL_ETH_RxCpltCallback (ETH_HandleTypeDef * heth)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 15.2.18 HAL\_ETH\_ErrorCallback

Function Name	<b>void HAL_ETH_ErrorCallback (ETH_HandleTypeDef * heth)</b>
Function Description	Ethernet transfer error callbacks.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 15.2.19 HAL\_ETH\_ReadPHYRegister

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ReadPHYRegister (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t * RegValue)</b>
Function Description	Reads a PHY register.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li><b>PHYReg</b>: PHY register address, is the index of one of the 32 PHY register. This parameter can be one of the following values: PHY_BCR: Transceiver Basic Control Register, PHY_BSR: Transceiver Basic Status Register. More PHY register could be read depending on the used PHY</li> <li><b>RegValue</b>: PHY register value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 15.2.20 HAL\_ETH\_WritePHYRegister

Function Name	<b>HAL_StatusTypeDef HAL_ETH_WritePHYRegister (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t RegValue)</b>
Function Description	Writes to a PHY register.



Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> <li>• <b>PHYReg</b>: PHY register address, is the index of one of the 32 PHY register. This parameter can be one of the following values: <code>PHY_BCR</code>: Transceiver Control Register. More PHY register could be written depending on the used PHY</li> <li>• <b>RegValue</b>: the value to write</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.21 HAL\_ETH\_Start

Function Name	<b>HAL_StatusTypeDef HAL_ETH_Start (ETH_HandleTypeDef * heth)</b>
Function Description	Enables Ethernet MAC and DMA reception/transmission.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.22 HAL\_ETH\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_ETH_Stop (ETH_HandleTypeDef * heth)</b>
Function Description	Stop Ethernet MAC and DMA reception/transmission.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.23 HAL\_ETH\_ConfigMAC

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ConfigMAC (ETH_HandleTypeDef * heth, ETH_MACInitTypeDef * macconf)</b>
Function Description	Set ETH MAC Configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> <li>• <b>macconf</b>: MAC Configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.24 HAL\_ETH\_ConfigDMA

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ConfigDMA (ETH_HandleTypeDef * heth, ETH_DMAInitTypeDef * dmaconf)</b>
Function Description	Sets ETH DMA Configuration.

Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li>• <b>dmaconf</b>: DMA Configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.25 HAL\_ETH\_GetState

Function Name	<b>HAL_ETH_StateTypeDef HAL_ETH_GetState (ETH_HandleTypeDef * heth)</b>
Function Description	Return the ETH HAL state.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

## 15.3 ETH Firmware driver defines

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

### 15.3.1 ETH

ETH

#### **ETH Address Aligned Beats**

ETH\_ADDRESSALIGNEDBEATS\_ENABLE

ETH\_ADDRESSALIGNEDBEATS\_DISABLE

#### **ETH Automatic Pad CRC Strip**

ETH\_AUTOMATICPADCRCSTRIP\_ENABLE

ETH\_AUTOMATICPADCRCSTRIP\_DISABLE

#### **ETH AutoNegotiation**

ETH\_AUTONEGOTIATION\_ENABLE

ETH\_AUTONEGOTIATION\_DISABLE

#### **ETH Back Off Limit**

ETH\_BACKOFFLIMIT\_10

ETH\_BACKOFFLIMIT\_8

ETH\_BACKOFFLIMIT\_4

ETH\_BACKOFFLIMIT\_1

#### **ETH Broadcast Frames Reception**

ETH\_BROADCASTFRAMESRECEPTION\_ENABLE

ETH\_BROADCASTFRAMESRECEPTION\_DISABLE

#### **ETH Buffers setting**

ETH\_MAX\_PACKET\_SIZE                      ETH\_HEADER + ETH\_EXTRA + ETH\_VLAN\_TAG +  
ETH\_MAX\_ETH\_PAYLOAD + ETH\_CRC

ETH_HEADER	6 byte Dest addr, 6 byte Src addr, 2 byte length/type
ETH_CRC	Ethernet CRC
ETH_EXTRA	Extra bytes in some cases
ETH_VLAN_TAG	optional 802.1q VLAN Tag
ETH_MIN_ETH_PAYLOAD	Minimum Ethernet payload size
ETH_MAX_ETH_PAYLOAD	Maximum Ethernet payload size
ETH_JUMBO_FRAME_PAYLOAD	Jumbo frame payload size
ETH_RX_BUF_SIZE	
ETH_RXBUFNB	
ETH_TX_BUF_SIZE	
ETH_TXBUFNB	
<b>ETH Carrier Sense</b>	
ETH_CARRIERSENCE_ENABLE	
ETH_CARRIERSENCE_DISABLE	
<b>ETH Checksum Mode</b>	
ETH_CHECKSUM_BY_HARDWARE	
ETH_CHECKSUM_BY_SOFTWARE	
<b>ETH Checksum Offload</b>	
ETH_CHECKSUMOFFLOAD_ENABLE	
ETH_CHECKSUMOFFLOAD_DISABLE	
<b>ETH Deferral Check</b>	
ETH_DEFFERRALCHECK_ENABLE	
ETH_DEFFERRALCHECK_DISABLE	
<b>ETH Destination Addr Filter</b>	
ETH_DESTINATIONADDRFILTER_NORMAL	
ETH_DESTINATIONADDRFILTER_INVERSE	
<b>ETH DMA Arbitration</b>	
ETH_DMAARBITRATION_ROUNDROBIN_RXTX_1_1	
ETH_DMAARBITRATION_ROUNDROBIN_RXTX_2_1	
ETH_DMAARBITRATION_ROUNDROBIN_RXTX_3_1	
ETH_DMAARBITRATION_ROUNDROBIN_RXTX_4_1	
ETH_DMAARBITRATION_RXPRIORTX	
<b>ETH DMA Flags</b>	
ETH_DMA_FLAG_TST	Time-stamp trigger interrupt (on DMA)
ETH_DMA_FLAG_PMT	PMT interrupt (on DMA)
ETH_DMA_FLAG_MMC	MMC interrupt (on DMA)

ETH_DMA_FLAG_DATATRANSFERERROR	Error bits 0-Rx DMA, 1-Tx DMA
ETH_DMA_FLAG_READWRITEERROR	Error bits 0-write trnsf, 1-read transfr
ETH_DMA_FLAG_ACCESSERROR	Error bits 0-data buffer, 1-desc. access
ETH_DMA_FLAG_NIS	Normal interrupt summary flag
ETH_DMA_FLAG_AIS	Abnormal interrupt summary flag
ETH_DMA_FLAG_ER	Early receive flag
ETH_DMA_FLAG_FBE	Fatal bus error flag
ETH_DMA_FLAG_ET	Early transmit flag
ETH_DMA_FLAG_RWT	Receive watchdog timeout flag
ETH_DMA_FLAG_RPS	Receive process stopped flag
ETH_DMA_FLAG_RBU	Receive buffer unavailable flag
ETH_DMA_FLAG_R	Receive flag
ETH_DMA_FLAG_TU	Underflow flag
ETH_DMA_FLAG_RO	Overflow flag
ETH_DMA_FLAG_TJT	Transmit jabber timeout flag
ETH_DMA_FLAG_TBU	Transmit buffer unavailable flag
ETH_DMA_FLAG_TPS	Transmit process stopped flag
ETH_DMA_FLAG_T	Transmit flag

**ETH DMA Interrupts**

ETH_DMA_IT_TST	Time-stamp trigger interrupt (on DMA)
ETH_DMA_IT_PMT	PMT interrupt (on DMA)
ETH_DMA_IT_MMC	MMC interrupt (on DMA)
ETH_DMA_IT_NIS	Normal interrupt summary
ETH_DMA_IT_AIS	Abnormal interrupt summary
ETH_DMA_IT_ER	Early receive interrupt
ETH_DMA_IT_FBE	Fatal bus error interrupt
ETH_DMA_IT_ET	Early transmit interrupt
ETH_DMA_IT_RWT	Receive watchdog timeout interrupt
ETH_DMA_IT_RPS	Receive process stopped interrupt
ETH_DMA_IT_RBU	Receive buffer unavailable interrupt
ETH_DMA_IT_R	Receive interrupt
ETH_DMA_IT_TU	Underflow interrupt
ETH_DMA_IT_RO	Overflow interrupt
ETH_DMA_IT_TJT	Transmit jabber timeout interrupt
ETH_DMA_IT_TBU	Transmit buffer unavailable interrupt
ETH_DMA_IT_TPS	Transmit process stopped interrupt

ETH\_DMA\_IT\_T Transmit interrupt

**ETH DMA overflow**

ETH\_DMA\_OVERFLOW\_RXFIFOCOUNTER Overflow bit for FIFO overflow counter

ETH\_DMA\_OVERFLOW\_MISSEDFRAMECOUNTER Overflow bit for missed frame counter

**ETH DMA receive process state**

ETH\_DMA\_RECEIVEPROCESS\_STOPPED Stopped - Reset or Stop Rx Command issued

ETH\_DMA\_RECEIVEPROCESS\_FETCHING Running - fetching the Rx descriptor

ETH\_DMA\_RECEIVEPROCESS\_WAITING Running - waiting for packet

ETH\_DMA\_RECEIVEPROCESS\_SUSPENDED Suspended - Rx Descriptor unavailable

ETH\_DMA\_RECEIVEPROCESS\_CLOSING Running - closing descriptor

ETH\_DMA\_RECEIVEPROCESS\_QUEUEING Running - queuing the receive frame into host memory

**ETH DMA RX Descriptor**

ETH\_DMARXDESC\_OWN OWN bit: descriptor is owned by DMA engine

ETH\_DMARXDESC\_AFM DA Filter Fail for the rx frame

ETH\_DMARXDESC\_FL Receive descriptor frame length

ETH\_DMARXDESC\_ES Error summary: OR of the following bits: DE || OE || IPC || LC || RWT || RE || CE

ETH\_DMARXDESC\_DE Descriptor error: no more descriptors for receive frame

ETH\_DMARXDESC\_SAF SA Filter Fail for the received frame

ETH\_DMARXDESC\_LE Frame size not matching with length field

ETH\_DMARXDESC\_OE Overflow Error: Frame was damaged due to buffer overflow

ETH\_DMARXDESC\_VLAN VLAN Tag: received frame is a VLAN frame

ETH\_DMARXDESC\_FS First descriptor of the frame

ETH\_DMARXDESC\_LS Last descriptor of the frame

ETH\_DMARXDESC\_IPV4HCE IPC Checksum Error: Rx Ipv4 header checksum error

ETH\_DMARXDESC\_LC Late collision occurred during reception

ETH\_DMARXDESC\_FT Frame type - Ethernet, otherwise 802.3

ETH\_DMARXDESC\_RWT Receive Watchdog Timeout: watchdog timer expired during reception

ETH\_DMARXDESC\_RE Receive error: error reported by MII interface

ETH\_DMARXDESC\_DBE Dribble bit error: frame contains non int multiple of 8 bits

ETH\_DMARXDESC\_CE CRC error

ETH\_DMARXDESC\_MAMPCE Rx MAC Address/Payload Checksum Error: Rx MAC address matched/ Rx Payload Checksum Error

ETH_DMARXDESC_DIC	Disable Interrupt on Completion
ETH_DMARXDESC_RBS2	Receive Buffer2 Size
ETH_DMARXDESC_RER	Receive End of Ring
ETH_DMARXDESC_RCH	Second Address Chained
ETH_DMARXDESC_RBS1	Receive Buffer1 Size
ETH_DMARXDESC_B1AP	Buffer1 Address Pointer
ETH_DMARXDESC_B2AP	Buffer2 Address Pointer

**ETH DMA Rx Descriptor Buffers**

ETH_DMARXDESC_BUFFER1	DMA Rx Desc Buffer1
ETH_DMARXDESC_BUFFER2	DMA Rx Desc Buffer2

**ETH DMA transmit process state**

ETH_DMA_TRANSMITPROCESS_STOPPED	Stopped - Reset or Stop Tx Command issued
ETH_DMA_TRANSMITPROCESS_FETCHING	Running - fetching the Tx descriptor
ETH_DMA_TRANSMITPROCESS_WAITING	Running - waiting for status
ETH_DMA_TRANSMITPROCESS_READING	Running - reading the data from host memory
ETH_DMA_TRANSMITPROCESS_SUSPENDED	Suspended - Tx Descriptor unavailable
ETH_DMA_TRANSMITPROCESS_CLOSING	Running - closing Rx descriptor

**ETH DMA TX Descriptor**

ETH_DMATXDESC_OWN	OWN bit: descriptor is owned by DMA engine
ETH_DMATXDESC_IC	Interrupt on Completion
ETH_DMATXDESC_LS	Last Segment
ETH_DMATXDESC_FS	First Segment
ETH_DMATXDESC_DC	Disable CRC
ETH_DMATXDESC_DP	Disable Padding
ETH_DMATXDESC_TTSE	Transmit Time Stamp Enable
ETH_DMATXDESC_CIC	Checksum Insertion Control: 4 cases
ETH_DMATXDESC_CIC_BYPASS	Do Nothing: Checksum Engine is bypassed
ETH_DMATXDESC_CIC_IPV4HEADER	IPV4 header Checksum Insertion
ETH_DMATXDESC_CIC_TCPUDPICMP_SEGMENT	TCP/UDP/ICMP Checksum Insertion calculated over segment only
ETH_DMATXDESC_CIC_TCPUDPICMP_FULL	TCP/UDP/ICMP Checksum Insertion fully calculated

ETH_DMATXDESC_TER	Transmit End of Ring
ETH_DMATXDESC_TCH	Second Address Chained
ETH_DMATXDESC_TTSS	Tx Time Stamp Status
ETH_DMATXDESC_IHE	IP Header Error
ETH_DMATXDESC_ES	Error summary: OR of the following bits: UE    ED    EC    LCO    NC    LCA    FF    JT
ETH_DMATXDESC_JT	Jabber Timeout
ETH_DMATXDESC_FF	Frame Flushed: DMA/MTL flushed the frame due to SW flush
ETH_DMATXDESC_PCE	Payload Checksum Error
ETH_DMATXDESC_LCA	Loss of Carrier: carrier lost during transmission
ETH_DMATXDESC_NC	No Carrier: no carrier signal from the transceiver
ETH_DMATXDESC_LCO	Late Collision: transmission aborted due to collision
ETH_DMATXDESC_EC	Excessive Collision: transmission aborted after 16 collisions
ETH_DMATXDESC_VF	VLAN Frame
ETH_DMATXDESC_CC	Collision Count
ETH_DMATXDESC_ED	Excessive Deferral
ETH_DMATXDESC_UF	Underflow Error: late data arrival from the memory
ETH_DMATXDESC_DB	Deferred Bit
ETH_DMATXDESC_TBS2	Transmit Buffer2 Size
ETH_DMATXDESC_TBS1	Transmit Buffer1 Size
ETH_DMATXDESC_B1AP	Buffer1 Address Pointer
ETH_DMATXDESC_B2AP	Buffer2 Address Pointer

#### ***ETH DMA Tx Descriptor Checksum Insertion Control***

ETH_DMATXDESC_CHECKSUMBYPASS	Checksum engine bypass
ETH_DMATXDESC_CHECKSUMIPV4HEADER	IPv4 header checksum insertion
ETH_DMATXDESC_CHECKSUMTCPUDPICMPSEGMENT	TCP/UDP/ICMP checksum insertion. Pseudo header checksum is assumed to be present
ETH_DMATXDESC_CHECKSUMTCPUDPICMPFULL	TCP/UDP/ICMP checksum fully in hardware including pseudo header

#### ***ETH DMA Tx Descriptor Segment***

ETH\_DMATXDESC\_LASTSEGMENTS    Last Segment

ETH\_DMATXDESC\_FIRSTSEGMENT    First Segment

**ETH Drop TCP IP Checksum Error Frame**

ETH\_DROPTCPIPCHECKSUMERRORFRAME\_ENABLE

ETH\_DROPTCPIPCHECKSUMERRORFRAME\_DISABLE

**ETH Duplex Mode**

ETH\_MODE\_FULLDUPLEX

ETH\_MODE\_HALFDUPLEX

**ETH Exported Macros**

\_\_HAL\_ETH\_RESET\_HANDLE\_STATE

**Description:**

- Reset ETH handle state.

**Parameters:**

- \_\_HANDLE\_\_: specifies the ETH handle.

**Return value:**

- None:

\_\_HAL\_ETH\_DMATXDESC\_GET\_FLAG

**Description:**

- Checks whether the specified ETHERNET DMA Tx Desc flag is set or not.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle
- \_\_FLAG\_\_: specifies the flag of TDES0 to check .

**Return value:**

- the: ETH\_DMATxDescFlag (SET or RESET).

\_\_HAL\_ETH\_DMARXDESC\_GET\_FLAG

**Description:**

- Checks whether the specified ETHERNET DMA Rx Desc flag is set or not.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle
- \_\_FLAG\_\_: specifies the flag of RDES0 to check.

**Return value:**

- the: ETH\_DMATxDescFlag (SET or RESET).

\_\_HAL\_ETH\_DMARXDESC\_ENABLE\_IT

**Description:**

- Enables the specified DMA Rx Desc



receive interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_DMARXDESC_DISABLE_IT`

**Description:**

- Disables the specified DMA Rx Desc receive interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_DMARXDESC_SET_OWN_BIT`

**Description:**

- Set the specified DMA Rx Desc Own bit.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_DMATXDESC_GET_COLLISION_COUNT`

**Description:**

- Returns the specified ETHERNET DMA Tx Desc collision count.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- The: Transmit descriptor collision counter value.

`__HAL_ETH_DMATXDESC_SET_OWN_BIT`

**Description:**

- Set the specified DMA Tx Desc Own bit.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_DMATXDESC_ENABLE_IT`

**Description:**

- Enables the specified DMA Tx Desc Transmit interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

`__HAL_ETH_DMATXDESC_DISABLE_IT`

**Return value:**

- None:

**Description:**

- Disables the specified DMA Tx Desc Transmit interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_DMATXDESC_CHECKSUM_INSERTION`

**Description:**

- Selects the specified ETHERNET DMA Tx Desc Checksum Insertion.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__CHECKSUM__`: specifies is the DMA Tx desc checksum insertion. This parameter can be one of the following values:
  - `ETH_DMATXDESC_CHECKSUMBYPASS`: Checksum bypass
  - `ETH_DMATXDESC_CHECKSUMIPv4HEADER`: IPv4 header checksum
  - `ETH_DMATXDESC_CHECKSUMTCPUDPICMPSEGMENT`: TCP/UDP/ICMP checksum. Pseudo header checksum is assumed to be present
  - `ETH_DMATXDESC_CHECKSUMTCPUDPICMPFULL`: TCP/UDP/ICMP checksum fully in hardware including pseudo header

**Return value:**

- None:

`__HAL_ETH_DMATXDESC_CRC_ENABLE`

**Description:**

- Enables the DMA Tx Desc CRC.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_DMATXDESC_CRC_DISABLE`

**Description:**

- Disables the DMA Tx Desc CRC.

**Parameters:**

`__HAL_ETH_DMATXDESC_SHORT_FRAME_PADDING_ENABLE`

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

**Description:**

- Enables the DMA Tx Desc padding for frame shorter than 64 bytes.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_DMATXDESC_SHORT_FRAME_PADDING_DISABLE`

**Description:**

- Disables the DMA Tx Desc padding for frame shorter than 64 bytes.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_MAC_ENABLE_IT`

**Description:**

- Enables the specified ETHERNET MAC interrupts.

**Parameters:**

- `__HANDLE__`: : ETH Handle
- `__INTERRUPT__`: specifies the ETHERNET MAC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - `ETH_MAC_IT_TST` : Time stamp trigger interrupt
  - `ETH_MAC_IT_PMT` : PMT interrupt

**Return value:**

- None:

`__HAL_ETH_MAC_DISABLE_IT`

**Description:**

- Disables the specified ETHERNET MAC interrupts.

**Parameters:**

- `__HANDLE__`: : ETH Handle
- `__INTERRUPT__`: specifies the ETHERNET MAC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:

- ETH\_MAC\_IT\_TST : Time stamp trigger interrupt
- ETH\_MAC\_IT\_PMT : PMT interrupt

**Return value:**

- None:

`__HAL_ETH_INITIATE_PAUSE_CONTROL_FRAME`

**Description:**

- Initiate a Pause Control Frame (Full-duplex only).

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_GET_FLOW_CONTROL_BUSY_STATUS`

**Description:**

- Checks whether the ETHERNET flow control busy bit is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- The: new state of flow control busy status bit (SET or RESET).

`__HAL_ETH_BACK_PRESSURE_ACTIVATION_ENABLE`

**Description:**

- Enables the MAC Back Pressure operation activation (Half-duplex only).

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_BACK_PRESSURE_ACTIVATION_DISABLE`

**Description:**

- Disables the MAC BackPressure operation activation (Half-duplex only).

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_MAC_GET_FLAG`

**Description:**

- Checks whether the specified ETHERNET MAC flag is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle

- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `ETH_MAC_FLAG_TST` : Time stamp trigger flag
  - `ETH_MAC_FLAG_MMCT` : MMC transmit flag
  - `ETH_MAC_FLAG_MMCR` : MMC receive flag
  - `ETH_MAC_FLAG_MMC` : MMC flag
  - `ETH_MAC_FLAG_PMT` : PMT flag

**Return value:**

- The: state of ETHERNET MAC flag.

**Description:**

- Enables the specified ETHERNET DMA interrupts.

**Parameters:**

- `__HANDLE__`: : ETH Handle
- `__INTERRUPT__`: specifies the ETHERNET DMA interrupt sources to be enabled

**Return value:**

- None:

**Description:**

- Disables the specified ETHERNET DMA interrupts.

**Parameters:**

- `__HANDLE__`: : ETH Handle
- `__INTERRUPT__`: specifies the ETHERNET DMA interrupt sources to be disabled.

**Return value:**

- None:

**Description:**

- Clears the ETHERNET DMA IT pending bit.

**Parameters:**

- `__HANDLE__`: : ETH Handle
- `__INTERRUPT__`: specifies the interrupt pending bit to clear.

**Return value:**

- None:

`__HAL_ETH_DMA_ENABLE_IT``__HAL_ETH_DMA_DISABLE_IT``__HAL_ETH_DMA_CLEAR_IT`

**\_\_HAL\_ETH\_DMA\_GET\_FLAG****Description:**

- Checks whether the specified ETHERNET DMA flag is set or not.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle
- \_\_FLAG\_\_: specifies the flag to check.

**Return value:**

- The: new state of ETH\_DMA\_FLAG (SET or RESET).

**\_\_HAL\_ETH\_DMA\_CLEAR\_FLAG****Description:**

- Checks whether the specified ETHERNET DMA flag is set or not.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle
- \_\_FLAG\_\_: specifies the flag to clear.

**Return value:**

- The: new state of ETH\_DMA\_FLAG (SET or RESET).

**\_\_HAL\_ETH\_GET\_DMA\_OVERFLOW\_STATUS****Description:**

- Checks whether the specified ETHERNET DMA overflow flag is set or not.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle
- \_\_OVERFLOW\_\_: specifies the DMA overflow flag to check. This parameter can be one of the following values:
  - ETH\_DMA\_OVERFLOW\_RXFIFO COUNTER : Overflow for FIFO Overflows Counter
  - ETH\_DMA\_OVERFLOW\_MISSED FRAMECOUNTER : Overflow for Buffer Unavailable Missed Frame Counter

**Return value:**

- The: state of ETHERNET DMA overflow Flag (SET or RESET).

**\_\_HAL\_ETH\_SET\_RECEIVE\_WATCHDOG\_TIMER****Description:**

- Set the DMA Receive status watchdog timer register value.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle
- \_\_VALUE\_\_: DMA Receive status

watchdog timer register value

**Return value:**

- None:

**Description:**

- Enables any unicast packet filtered by the MAC address recognition to be a wake-up frame.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None:

**Description:**

- Disables any unicast packet filtered by the MAC address recognition to be a wake-up frame.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None:

**Description:**

- Enables the MAC Wake-Up Frame Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None:

**Description:**

- Disables the MAC Wake-Up Frame Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None:

**Description:**

- Enables the MAC Magic Packet Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

`__HAL_ETH_GLOBAL_UNICAST_WAKEUP_ENABLE`

`__HAL_ETH_GLOBAL_UNICAST_WAKEUP_DISABLE`

`__HAL_ETH_WAKEUP_FRAME_DETECTION_ENABLE`

`__HAL_ETH_WAKEUP_FRAME_DETECTION_DISABLE`

`__HAL_ETH_MAGIC_PACKET_DETECTION_ENABLE`

`__HAL_ETH_MAGIC_PACKET_DETECTI  
ON_DISABLE`

**Return value:**

- None:

**Description:**

- Disables the MAC Magic Packet Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None:

`__HAL_ETH_POWER_DOWN_ENABLE`

**Description:**

- Enables the MAC Power Down.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_POWER_DOWN_DISABLE`

**Description:**

- Disables the MAC Power Down.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None:

`__HAL_ETH_GET_PMT_FLAG_STATUS`

**Description:**

- Checks whether the specified ETHERNET PMT flag is set or not.

**Parameters:**

- `__HANDLE__`: ETH Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `ETH_PMT_FLAG_WUFFRPR` : Wake-Up Frame Filter Register Pointer Reset
  - `ETH_PMT_FLAG_WUFR` : Wake-Up Frame Received
  - `ETH_PMT_FLAG_MPR` : Magic Packet Received

**Return value:**

- The: new state of ETHERNET PMT Flag (SET or RESET).

`__HAL_ETH_MMC_COUNTER_FULL_PR`

**Description:**



ESET	<ul style="list-style-type: none"> <li>Preset and Initialize the MMC counters to almost-full value: 0xFFFF_FFF0 (full - 16)</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>__HANDLE__: ETH Handle.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None:</li> </ul>
__HAL_ETH_MMC_COUNTER_HALF_PRESET	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Preset and Initialize the MMC counters to almost-half value: 0x7FFF_FFF0 (half - 16)</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>__HANDLE__: ETH Handle.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None:</li> </ul>
__HAL_ETH_MMC_COUNTER_FREEZE_ENABLE	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Enables the MMC Counter Freeze.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>__HANDLE__: ETH Handle.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None:</li> </ul>
__HAL_ETH_MMC_COUNTER_FREEZE_DISABLE	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Disables the MMC Counter Freeze.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>__HANDLE__: ETH Handle.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None:</li> </ul>
__HAL_ETH_ETH_MMC_RESET_ONREAD_ENABLE	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Enables the MMC Reset On Read.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>__HANDLE__: ETH Handle.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None:</li> </ul>
__HAL_ETH_ETH_MMC_RESET_ONREAD_DISABLE	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Disables the MMC Reset On Read.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>__HANDLE__: ETH Handle.</li> </ul>

`__HAL_ETH_ETH_MMC_COUNTER_ROLLOVER_ENABLE`

**Return value:**

- None:

**Description:**

- Enables the MMC Counter Stop Rollover.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None:

`__HAL_ETH_ETH_MMC_COUNTER_ROLLOVER_DISABLE`

**Description:**

- Disables the MMC Counter Stop Rollover.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None:

`__HAL_ETH_MMC_COUNTERS_RESET`

**Description:**

- Resets the MMC Counters.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None:

`__HAL_ETH_MMC_RX_IT_ENABLE`

**Description:**

- Enables the specified ETHERNET MMC Rx interrupts.

**Parameters:**

- `__HANDLE__`: ETH Handle.
- `__INTERRUPT__`: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - `ETH_MMC_IT_RGUF` : When Rx good unicast frames counter reaches half the maximum value
  - `ETH_MMC_IT_RFAE` : When Rx alignment error counter reaches half the maximum value
  - `ETH_MMC_IT_RFCE` : When Rx crc error counter reaches half the maximum value

**Return value:**

**\_\_HAL\_ETH\_MMC\_RX\_IT\_DISABLE**

- None:

**Description:**

- Disables the specified ETHERNET MMC Rx interrupts.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle.
- \_\_INTERRUPT\_\_: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - ETH\_MMC\_IT\_RGUF : When Rx good unicast frames counter reaches half the maximum value
  - ETH\_MMC\_IT\_RFAE : When Rx alignment error counter reaches half the maximum value
  - ETH\_MMC\_IT\_RFCE : When Rx crc error counter reaches half the maximum value

**Return value:**

- None:

**\_\_HAL\_ETH\_MMC\_TX\_IT\_ENABLE****Description:**

- Enables the specified ETHERNET MMC Tx interrupts.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle.
- \_\_INTERRUPT\_\_: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - ETH\_MMC\_IT\_TGF : When Tx good frame counter reaches half the maximum value
  - ETH\_MMC\_IT\_TGFMSC: When Tx good multi col counter reaches half the maximum value
  - ETH\_MMC\_IT\_TGFSC : When Tx good single col counter reaches half the maximum value

**Return value:**

- None:

**\_\_HAL\_ETH\_MMC\_TX\_IT\_DISABLE****Description:**

- Disables the specified ETHERNET MMC Tx interrupts.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle.

- `__INTERRUPT__`: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - `ETH_MMC_IT_TGF` : When Tx good frame counter reaches half the maximum value
  - `ETH_MMC_IT_TGFMSC`: When Tx good multi col counter reaches half the maximum value
  - `ETH_MMC_IT_TGFSC` : When Tx good single col counter reaches half the maximum value

**Return value:**

- None:

`__HAL_ETH_WAKEUP_EXTI_ENABLE_I`  
`T`

**Description:**

- Enables the ETH External interrupt line.

**Return value:**

- None:

`__HAL_ETH_WAKEUP_EXTI_DISABLE_I`  
`T`

**Description:**

- Disables the ETH External interrupt line.

**Return value:**

- None:

`__HAL_ETH_WAKEUP_EXTI_ENABLE_E`  
`VENT`

**Description:**

- Enable event on ETH External event line.

**Return value:**

- None.:

`__HAL_ETH_WAKEUP_EXTI_DISABLE_`  
`EVENT`

**Description:**

- Disable event on ETH External event line.

**Return value:**

- None.:

`__HAL_ETH_WAKEUP_EXTI_GET_FLAG`

**Description:**

- Get flag of the ETH External interrupt line.

**Return value:**

- None:

`__HAL_ETH_WAKEUP_EXTI_CLEAR_FL`  
`AG`

**Description:**

- Clear flag of the ETH External interrupt line.

`__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_EDGE_TRIGGER`

**Return value:**

- None:

**Description:**

- Enables rising edge trigger to the ETH External interrupt line.

**Return value:**

- None:

`__HAL_ETH_WAKEUP_EXTI_DISABLE_RISING_EDGE_TRIGGER`

**Description:**

- Disables the rising edge trigger to the ETH External interrupt line.

**Return value:**

- None:

`__HAL_ETH_WAKEUP_EXTI_ENABLE_FALLING_EDGE_TRIGGER`

**Description:**

- Enables falling edge trigger to the ETH External interrupt line.

**Return value:**

- None:

`__HAL_ETH_WAKEUP_EXTI_DISABLE_FALLING_EDGE_TRIGGER`

**Description:**

- Disables falling edge trigger to the ETH External interrupt line.

**Return value:**

- None:

`__HAL_ETH_WAKEUP_EXTI_ENABLE_FALLINGRISING_TRIGGER`

**Description:**

- Enables rising/falling edge trigger to the ETH External interrupt line.

**Return value:**

- None:

`__HAL_ETH_WAKEUP_EXTI_DISABLE_FALLINGRISING_TRIGGER`

**Description:**

- Disables rising/falling edge trigger to the ETH External interrupt line.

**Return value:**

- None:

`__HAL_ETH_WAKEUP_EXTI_GENERATE_SWIT`

**Description:**

- Generate a Software interrupt on selected EXTI line.

**Return value:**

- None.:

**ETH EXTI LINE WAKEUP**

ETH\_EXTI\_LINE\_WAKEUP External interrupt line 19 Connected to the ETH EXTI Line

**ETH Fixed Burst**

ETH\_FIXEDBURST\_ENABLE

ETH\_FIXEDBURST\_DISABLE

**ETH Flush Received Frame**

ETH\_FLUSHRECEIVEDFRAME\_ENABLE

ETH\_FLUSHRECEIVEDFRAME\_DISABLE

**ETH Forward Error Frames**

ETH\_FORWARDERRORFRAMES\_ENABLE

ETH\_FORWARDERRORFRAMES\_DISABLE

**ETH Forward Undersized Good Frames**

ETH\_FORWARDUNDERSIZEDGOODFRAMES\_ENABLE

ETH\_FORWARDUNDERSIZEDGOODFRAMES\_DISABLE

**ETH Inter Frame Gap**

ETH\_INTERFRAMEGAP\_96BIT minimum IFG between frames during transmission is 96Bit

ETH\_INTERFRAMEGAP\_88BIT minimum IFG between frames during transmission is 88Bit

ETH\_INTERFRAMEGAP\_80BIT minimum IFG between frames during transmission is 80Bit

ETH\_INTERFRAMEGAP\_72BIT minimum IFG between frames during transmission is 72Bit

ETH\_INTERFRAMEGAP\_64BIT minimum IFG between frames during transmission is 64Bit

ETH\_INTERFRAMEGAP\_56BIT minimum IFG between frames during transmission is 56Bit

ETH\_INTERFRAMEGAP\_48BIT minimum IFG between frames during transmission is 48Bit

ETH\_INTERFRAMEGAP\_40BIT minimum IFG between frames during transmission is 40Bit

**ETH Jabber**

ETH\_JABBER\_ENABLE

ETH\_JABBER\_DISABLE

**ETH Loop Back Mode**

ETH\_LOOPBACKMODE\_ENABLE

ETH\_LOOPBACKMODE\_DISABLE

**ETH MAC addresses**

ETH\_MAC\_ADDRESS0

ETH\_MAC\_ADDRESS1

ETH\_MAC\_ADDRESS2

ETH\_MAC\_ADDRESS3

***ETH\_MAC Addresses Filter Mask Bytes***

ETH\_MAC\_ADDRESSMASK\_BYTE6 Mask MAC Address high reg bits [15:8]

ETH\_MAC\_ADDRESSMASK\_BYTE5 Mask MAC Address high reg bits [7:0]

ETH\_MAC\_ADDRESSMASK\_BYTE4 Mask MAC Address low reg bits [31:24]

ETH\_MAC\_ADDRESSMASK\_BYTE3 Mask MAC Address low reg bits [23:16]

ETH\_MAC\_ADDRESSMASK\_BYTE2 Mask MAC Address low reg bits [15:8]

ETH\_MAC\_ADDRESSMASK\_BYTE1 Mask MAC Address low reg bits [7:0]

***ETH MAC Addresses Filter SA DA***

ETH\_MAC\_ADDRESSFILTER\_SA

ETH\_MAC\_ADDRESSFILTER\_DA

***ETH MAC Debug Flags***

ETH\_MAC\_TXFIFO\_FULL

ETH\_MAC\_TXFIFONOT\_EMPTY

ETH\_MAC\_TXFIFO\_WRITE\_ACTIVE

ETH\_MAC\_TXFIFO\_IDLE

ETH\_MAC\_TXFIFO\_READ

ETH\_MAC\_TXFIFO\_WAITING

ETH\_MAC\_TXFIFO\_WRITING

ETH\_MAC\_TRANSMISSION\_PAUSE

ETH\_MAC\_TRANSMITFRAMECONTROLLER\_IDLE

ETH\_MAC\_TRANSMITFRAMECONTROLLER\_WAITING

ETH\_MAC\_TRANSMITFRAMECONTROLLER\_GENERATING\_PCF

ETH\_MAC\_TRANSMITFRAMECONTROLLER\_TRANSFERRING

ETH\_MAC\_MII\_TRANSMIT\_ACTIVE

ETH\_MAC\_RXFIFO\_EMPTY

ETH\_MAC\_RXFIFO\_BELOW\_THRESHOLD

ETH\_MAC\_RXFIFO\_ABOVE\_THRESHOLD

ETH\_MAC\_RXFIFO\_FULL

ETH\_MAC\_READCONTROLLER\_IDLE

ETH\_MAC\_READCONTROLLER\_READING\_DATA

ETH\_MAC\_READCONTROLLER\_READING\_STATUS

ETH\_MAC\_READCONTROLLER\_

ETH\_MAC\_RXFIFO\_WRITE\_ACTIVE

ETH\_MAC\_SMALL\_FIFO\_NOTACTIVE

ETH\_MAC\_SMALL\_FIFO\_READ\_ACTIVE  
ETH\_MAC\_SMALL\_FIFO\_WRITE\_ACTIVE  
ETH\_MAC\_SMALL\_FIFO\_RW\_ACTIVE  
ETH\_MAC\_MII\_RECEIVE\_PROTOCOL\_ACTIVE

**ETH MAC Flags**

ETH\_MAC\_FLAG\_TST      Time stamp trigger flag (on MAC)  
ETH\_MAC\_FLAG\_MMCT    MMC transmit flag  
ETH\_MAC\_FLAG\_MMCR    MMC receive flag  
ETH\_MAC\_FLAG\_MMC      MMC flag (on MAC)  
ETH\_MAC\_FLAG\_PMT      PMT flag (on MAC)

**ETH MAC Interrupts**

ETH\_MAC\_IT\_TST      Time stamp trigger interrupt (on MAC)  
ETH\_MAC\_IT\_MMCT    MMC transmit interrupt  
ETH\_MAC\_IT\_MMCR    MMC receive interrupt  
ETH\_MAC\_IT\_MMC      MMC interrupt (on MAC)  
ETH\_MAC\_IT\_PMT      PMT interrupt (on MAC)

**ETH Media Interface**

ETH\_MEDIA\_INTERFACE\_MII  
ETH\_MEDIA\_INTERFACE\_RMII

**ETH MMC Rx Interrupts**

ETH\_MMC\_IT\_RGUF    When Rx good unicast frames counter reaches half the maximum value  
ETH\_MMC\_IT\_RFAE    When Rx alignment error counter reaches half the maximum value  
ETH\_MMC\_IT\_RFCE    When Rx crc error counter reaches half the maximum value

**ETH MMC Tx Interrupts**

ETH\_MMC\_IT\_TGF      When Tx good frame counter reaches half the maximum value  
ETH\_MMC\_IT\_TGFMSC   When Tx good multi col counter reaches half the maximum value  
ETH\_MMC\_IT\_TGFSC    When Tx good single col counter reaches half the maximum value

**ETH Multicast Frames Filter**

ETH\_MULTICASTFRAMESFILTER\_PERFECTHASHTABLE  
ETH\_MULTICASTFRAMESFILTER\_HASHTABLE  
ETH\_MULTICASTFRAMESFILTER\_PERFECT  
ETH\_MULTICASTFRAMESFILTER\_NONE

**ETH Pass Control Frames**

ETH\_PASSCONTROLFRAMES\_BLOCKALL

MAC filters all  
control frames from



	reaching the application
ETH_PASSCONTROLFRAMES_FORWARDALL	MAC forwards all control frames to application even if they fail the Address Filter
ETH_PASSCONTROLFRAMES_FORWARDPASSEDDADDRFILTER	MAC forwards control frames that pass the Address Filter.

**ETH Pause Low Threshold**

ETH_PAUSELOWTHRESHOLD_MINUS4	Pause time minus 4 slot times
ETH_PAUSELOWTHRESHOLD_MINUS28	Pause time minus 28 slot times
ETH_PAUSELOWTHRESHOLD_MINUS144	Pause time minus 144 slot times
ETH_PAUSELOWTHRESHOLD_MINUS256	Pause time minus 256 slot times

**ETH PMT Flags**

ETH_PMT_FLAG_WUFFRPR	Wake-Up Frame Filter Register Pointer Reset
ETH_PMT_FLAG_WUFR	Wake-Up Frame Received
ETH_PMT_FLAG_MPR	Magic Packet Received

**ETH Private Constants**

LINKED\_STATE\_TIMEOUT\_VALUE  
 AUTONEGO\_COMPLETED\_TIMEOUT\_VALUE  
 ETH\_REG\_WRITE\_DELAY  
 ETH\_SUCCESS  
 ETH\_ERROR  
 ETH\_DMATXDESC\_COLLISION\_COUNTSHIFT  
 ETH\_DMATXDESC\_BUFFER2\_SIZESHIFT  
 ETH\_DMARXDESC\_FRAME\_LENGTHSHIFT  
 ETH\_DMARXDESC\_BUFFER2\_SIZESHIFT  
 ETH\_DMARXDESC\_FRAMELENGTHSHIFT  
 ETH\_MAC\_ADDR\_HBASE  
 ETH\_MAC\_ADDR\_LBASE  
 ETH\_MACMIAR\_CR\_MASK  
 ETH\_MACCR\_CLEAR\_MASK  
 ETH\_MACFCR\_CLEAR\_MASK  
 ETH\_DMAOMR\_CLEAR\_MASK  
 ETH\_WAKEUP\_REGISTER\_LENGTH  
 ETH\_DMA\_RX\_OVERFLOW\_MISSEDFRAMES\_COUNTERSHIFT

***ETH\_Private\_Macros***

IS\_ETH\_PHY\_ADDRESS  
IS\_ETH\_AUTONEGOTIATION  
IS\_ETH\_SPEED  
IS\_ETH\_DUPLEX\_MODE  
IS\_ETH\_DUPLEX\_MODE  
IS\_ETH\_RX\_MODE  
IS\_ETH\_RX\_MODE  
IS\_ETH\_RX\_MODE  
IS\_ETH\_CHECKSUM\_MODE  
IS\_ETH\_MEDIA\_INTERFACE  
IS\_ETH\_WATCHDOG  
IS\_ETH\_JABBER  
IS\_ETH\_INTER\_FRAME\_GAP  
IS\_ETH\_CARRIER\_SENSE  
IS\_ETH\_RECEIVE\_OWN  
IS\_ETH\_LOOPBACK\_MODE  
IS\_ETH\_CHECKSUM\_OFFLOAD  
IS\_ETH\_RETRY\_TRANSMISSION  
IS\_ETH\_AUTOMATIC\_PADCRC\_STRIP  
IS\_ETH\_BACKOFF\_LIMIT  
IS\_ETH\_DEFERRAL\_CHECK  
IS\_ETH\_RECEIVE\_ALL  
IS\_ETH\_SOURCE\_ADDR\_FILTER  
IS\_ETH\_CONTROL\_FRAMES  
IS\_ETH\_BROADCAST\_FRAMES\_RECEPTION  
IS\_ETH\_DESTINATION\_ADDR\_FILTER  
IS\_ETH\_PROMISCUOUS\_MODE  
IS\_ETH\_MULTICAST\_FRAMES\_FILTER  
IS\_ETH\_UNICAST\_FRAMES\_FILTER  
IS\_ETH\_PAUSE\_TIME  
IS\_ETH\_ZEROQUANTA\_PAUSE  
IS\_ETH\_PAUSE\_LOW\_THRESHOLD  
IS\_ETH\_UNICAST\_PAUSE\_FRAME\_DETECT  
IS\_ETH\_RECEIVE\_FLOWCONTROL  
IS\_ETH\_TRANSMIT\_FLOWCONTROL

IS\_ETH\_VLAN\_TAG\_COMPARISON  
IS\_ETH\_VLAN\_TAG\_IDENTIFIER  
IS\_ETH\_MAC\_ADDRESS0123  
IS\_ETH\_MAC\_ADDRESS123  
IS\_ETH\_MAC\_ADDRESS\_FILTER  
IS\_ETH\_MAC\_ADDRESS\_MASK  
IS\_ETH\_DROP\_TCPIP\_CHECKSUM\_FRAME  
IS\_ETH\_RECEIVE\_STORE\_FORWARD  
IS\_ETH\_FLUSH\_RECEIVE\_FRAME  
IS\_ETH\_TRANSMIT\_STORE\_FORWARD  
IS\_ETH\_TRANSMIT\_THRESHOLD\_CONTROL  
IS\_ETH\_FORWARD\_ERROR\_FRAMES  
IS\_ETH\_FORWARD\_UNDERSIZED\_GOOD\_FRAMES  
IS\_ETH\_RECEIVE\_THRESHOLD\_CONTROL  
IS\_ETH\_SECOND\_FRAME\_OPERATE  
IS\_ETH\_ADDRESS\_ALIGNED\_BEATS  
IS\_ETH\_FIXED\_BURST  
IS\_ETH\_RXDMA\_BURST\_LENGTH  
IS\_ETH\_TXDMA\_BURST\_LENGTH  
IS\_ETH\_DMA\_DESC\_SKIP\_LENGTH  
IS\_ETH\_DMA\_ARBITRATION\_ROUNDROBIN\_RXTX  
IS\_ETH\_DMA\_TXDESC\_SEGMENT  
IS\_ETH\_DMA\_TXDESC\_CHECKSUM  
IS\_ETH\_DMATXDESC\_BUFFER\_SIZE  
IS\_ETH\_DMA\_RXDESC\_BUFFER  
IS\_ETH\_DMA\_GET\_OVERFLOW

***ETH Promiscuous Mode***

ETH\_PROMISCUOUS\_MODE\_ENABLE  
ETH\_PROMISCUOUS\_MODE\_DISABLE

***ETH Receive All***

ETH\_RECEIVEALL\_ENABLE  
ETH\_RECEIVEALL\_DISABLE

***ETH Receive Flow Control***

ETH\_RECEIVEFLOWCONTROL\_ENABLE  
ETH\_RECEIVEFLOWCONTROL\_DISABLE

***ETH Receive Own***

ETH\_RECEIVEOWN\_ENABLE

ETH\_RECEIVEOWN\_DISABLE

***ETH Receive Store Forward***

ETH\_RECEIVESTOREFORWARD\_ENABLE

ETH\_RECEIVESTOREFORWARD\_DISABLE

***ETH Receive Threshold Control***

ETH\_RECEIVEDTHRESHOLDCONTROL\_64BYTES      threshold level of the MTL  
Receive FIFO is 64 Bytes

ETH\_RECEIVEDTHRESHOLDCONTROL\_32BYTES      threshold level of the MTL  
Receive FIFO is 32 Bytes

ETH\_RECEIVEDTHRESHOLDCONTROL\_96BYTES      threshold level of the MTL  
Receive FIFO is 96 Bytes

ETH\_RECEIVEDTHRESHOLDCONTROL\_128BYTES      threshold level of the MTL  
Receive FIFO is 128 Bytes

***ETH Retry Transmission***

ETH\_RETRYTRANSMISSION\_ENABLE

ETH\_RETRYTRANSMISSION\_DISABLE

***ETH Rx DMA Burst Length***

ETH\_RXDMABURSTLENGTH\_1BEAT      maximum number of beats to be  
transferred in one RxDMA transaction  
is 1

ETH\_RXDMABURSTLENGTH\_2BEAT      maximum number of beats to be  
transferred in one RxDMA transaction  
is 2

ETH\_RXDMABURSTLENGTH\_4BEAT      maximum number of beats to be  
transferred in one RxDMA transaction  
is 4

ETH\_RXDMABURSTLENGTH\_8BEAT      maximum number of beats to be  
transferred in one RxDMA transaction  
is 8

ETH\_RXDMABURSTLENGTH\_16BEAT      maximum number of beats to be  
transferred in one RxDMA transaction  
is 16

ETH\_RXDMABURSTLENGTH\_32BEAT      maximum number of beats to be  
transferred in one RxDMA transaction  
is 32

ETH\_RXDMABURSTLENGTH\_4XPBL\_4BEAT      maximum number of beats to be  
transferred in one RxDMA transaction  
is 4

ETH\_RXDMABURSTLENGTH\_4XPBL\_8BEAT      maximum number of beats to be  
transferred in one RxDMA transaction  
is 8

ETH\_RXDMABURSTLENGTH\_4XPBL\_16BEAT      maximum number of beats to be  
transferred in one RxDMA transaction

	is 16
ETH_RXDMABURSTLENGTH_4XPBL_32BEAT	maximum number of beats to be transferred in one RxDMA transaction is 32
ETH_RXDMABURSTLENGTH_4XPBL_64BEAT	maximum number of beats to be transferred in one RxDMA transaction is 64
ETH_RXDMABURSTLENGTH_4XPBL_128BEAT	maximum number of beats to be transferred in one RxDMA transaction is 128

**ETH Rx Mode**

ETH\_RXPOLLING\_MODE

ETH\_RXINTERRUPT\_MODE

**ETH Second Frame Operate**

ETH\_SECONDFRAMEOPERARTE\_ENABLE

ETH\_SECONDFRAMEOPERARTE\_DISABLE

**ETH Source Addr Filter**

ETH\_SOURCEADDRFILTER\_NORMAL\_ENABLE

ETH\_SOURCEADDRFILTER\_INVERSE\_ENABLE

ETH\_SOURCEADDRFILTER\_DISABLE

**ETH Speed**

ETH\_SPEED\_10M

ETH\_SPEED\_100M

**ETH Transmit Flow Control**

ETH\_TRANSMITFLOWCONTROL\_ENABLE

ETH\_TRANSMITFLOWCONTROL\_DISABLE

**ETH Transmit Store Forward**

ETH\_TRANSMITSTOREFORWARD\_ENABLE

ETH\_TRANSMITSTOREFORWARD\_DISABLE

**ETH Transmit Threshold Control**

ETH_TRANSMITTHRESHOLDCONTROL_64BYTES	threshold level of the MTL Transmit FIFO is 64 Bytes
--------------------------------------	--

ETH_TRANSMITTHRESHOLDCONTROL_128BYTES	threshold level of the MTL Transmit FIFO is 128 Bytes
---------------------------------------	---

ETH_TRANSMITTHRESHOLDCONTROL_192BYTES	threshold level of the MTL Transmit FIFO is 192 Bytes
---------------------------------------	---

ETH_TRANSMITTHRESHOLDCONTROL_256BYTES	threshold level of the MTL Transmit FIFO is 256 Bytes
---------------------------------------	---

ETH_TRANSMITTHRESHOLDCONTROL_40BYTES	threshold level of the MTL Transmit FIFO is 40 Bytes
--------------------------------------	--

ETH_TRANSMITTHRESHOLDCONTROL_32BYTES	threshold level of the MTL Transmit FIFO is 32 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_24BYTES	threshold level of the MTL Transmit FIFO is 24 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_16BYTES	threshold level of the MTL Transmit FIFO is 16 Bytes

**ETH Tx DMA Burst Length**

ETH_TXDMABURSTLENGTH_1BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 1
ETH_TXDMABURSTLENGTH_2BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 2
ETH_TXDMABURSTLENGTH_4BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 4
ETH_TXDMABURSTLENGTH_8BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 8
ETH_TXDMABURSTLENGTH_16BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 16
ETH_TXDMABURSTLENGTH_32BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 32
ETH_TXDMABURSTLENGTH_4XPBL_4BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 4
ETH_TXDMABURSTLENGTH_4XPBL_8BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 8
ETH_TXDMABURSTLENGTH_4XPBL_16BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 16
ETH_TXDMABURSTLENGTH_4XPBL_32BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 32
ETH_TXDMABURSTLENGTH_4XPBL_64BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 64
ETH_TXDMABURSTLENGTH_4XPBL_128BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 128

**ETH Unicast Frames Filter**

ETH_UNICASTFRAMESFILTER_PERFECTHASHTABLE
ETH_UNICASTFRAMESFILTER_HASHTABLE

ETH\_UNICASTFRAMESFILTER\_PERFECT

***ETH Unicast Pause Frame Detect***

ETH\_UNICASTPAUSEFRAMEDETECT\_ENABLE

ETH\_UNICASTPAUSEFRAMEDETECT\_DISABLE

***ETH VLAN Tag Comparison***

ETH\_VLANTAGCOMPARISON\_12BIT

ETH\_VLANTAGCOMPARISON\_16BIT

***ETH Watchdog***

ETH\_WATCHDOG\_ENABLE

ETH\_WATCHDOG\_DISABLE

***ETH Zero Quanta Pause***

ETH\_ZEROQUANTAPAUSE\_ENABLE

ETH\_ZEROQUANTAPAUSE\_DISABLE

## 16 HAL FLASH Generic Driver

### 16.1 FLASH Firmware driver registers structures

#### 16.1.1 FLASH\_ProcessTypeDef

*FLASH\_ProcessTypeDef* is defined in the stm32f1xx\_hal\_flash.h

##### Data Fields

- *\_\_IO FLASH\_ProcedureTypeDef ProcedureOnGoing*
- *\_\_IO uint32\_t DataRemaining*
- *\_\_IO uint32\_t Address*
- *\_\_IO uint64\_t Data*
- *HAL\_LockTypeDef Lock*
- *\_\_IO uint32\_t ErrorCode*

##### Field Documentation

- *\_\_IO FLASH\_ProcedureTypeDef FLASH\_ProcessTypeDef::ProcedureOnGoing*
- *\_\_IO uint32\_t FLASH\_ProcessTypeDef::DataRemaining*
- *\_\_IO uint32\_t FLASH\_ProcessTypeDef::Address*
- *\_\_IO uint64\_t FLASH\_ProcessTypeDef::Data*
- *HAL\_LockTypeDef FLASH\_ProcessTypeDef::Lock*
- *\_\_IO uint32\_t FLASH\_ProcessTypeDef::ErrorCode*

### 16.2 FLASH Firmware driver API description

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

#### 16.2.1 FLASH peripheral features

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

The Flash memory interface accelerates code execution with a system of instruction prefetch.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Prefetch on I-Code
- Option Bytes programming

#### 16.2.2 How to use this driver



This driver provides functions and macros to configure and program the FLASH memory of all STM32F1xx devices. These functions are split in 3 groups:

1. FLASH Memory I/O Programming functions: this group includes all needed functions to erase and program the main memory:
  - Lock and Unlock the FLASH interface
  - Erase function: Erase page, erase all pages
  - Program functions: half word, word and doubleword
2. Option Bytes Programming functions: this group includes all needed functions to manage the Option Bytes:
  - Lock and Unlock the Option Bytes
  - Erase Option Bytes
  - Set/Reset the write protection
  - Set the Read protection Level
  - Program the user Option Bytes
  - Program the data Option Bytes
  - Launch the Option Bytes loader
3. Interrupts and flags management functions : this group includes all needed functions to:
  - Handle FLASH interrupts
  - Wait for last FLASH operation according to its status
  - Get error flag status

In addition to these function, this driver includes a set of macros allowing to handle the following operations:

- Set the latency
- Enable/Disable the prefetch buffer
- Enable/Disable the half cycle access
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

### 16.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations (write/erase).

- [\*HAL\\_FLASH\\_Program\(\)\*](#)
- [\*HAL\\_FLASH\\_Program\\_IT\(\)\*](#)
- [\*HAL\\_FLASH\\_IRQHandler\(\)\*](#)
- [\*HAL\\_FLASH\\_EndOfOperationCallback\(\)\*](#)
- [\*HAL\\_FLASH\\_OperationErrorCallback\(\)\*](#)

### 16.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

- [\*HAL\\_FLASH\\_Unlock\(\)\*](#)
- [\*HAL\\_FLASH\\_Lock\(\)\*](#)
- [\*HAL\\_FLASH\\_OB\\_Unlock\(\)\*](#)
- [\*HAL\\_FLASH\\_OB\\_Lock\(\)\*](#)
- [\*HAL\\_FLASH\\_OB\\_Launch\(\)\*](#)

## 16.2.5 Peripheral State functions

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

- [HAL\\_FLASH\\_GetError\(\)](#)

## 16.2.6 HAL\_FLASH\_Program

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint64_t Data)</b>
Function Description	Program halfword, word or double word at a specified address.
Parameters	<ul style="list-style-type: none"> <li>• <b>TypeProgram:</b> Indicate the way to program at a specified address. This parameter can be a value of Type Program</li> <li>• <b>Address:</b> Specifies the address to be programmed.</li> <li>• <b>Data:</b> Specifies the data to be programmed</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL_StatusTypeDef HAL Status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interface</li> <li>• If an erase and a program operations are requested simultaneously, the erase operation is performed before the program one.</li> <li>• FLASH should be previously erased before new programming (only exception to this is when 0x0000 is programmed)</li> </ul>

## 16.2.7 HAL\_FLASH\_Program\_IT

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t Data)</b>
Function Description	Program halfword, word or double word at a specified address with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>TypeProgram:</b> Indicate the way to program at a specified address. This parameter can be a value of Type Program</li> <li>• <b>Address:</b> Specifies the address to be programmed.</li> <li>• <b>Data:</b> Specifies the data to be programmed</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL_StatusTypeDef HAL Status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interface</li> <li>• If an erase and a program operations are requested simultaneously, the erase operation is performed before the program one.</li> </ul>

**16.2.8 HAL\_FLASH\_IRQHandler**

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

**16.2.9 HAL\_FLASH\_EndOfOperationCallback**

Function Name	<b>void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)</b>
Function Description	FLASH end of operation interrupt callback.
Parameters	<ul style="list-style-type: none"> <li><b>ReturnValue:</b> The value saved in this parameter depends on the ongoing procedure Mass Erase: No return value expectedPages Erase: Address of the page which has been erased Program: Address which was selected for data program</li> </ul>
Return values	<ul style="list-style-type: none"> <li>none</li> </ul>

**16.2.10 HAL\_FLASH\_OperationErrorCallback**

Function Name	<b>void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)</b>
Function Description	FLASH operation error interrupt callback.
Parameters	<ul style="list-style-type: none"> <li><b>ReturnValue:</b> The value saved in this parameter depends on the ongoing procedure Mass Erase: No return value expectedPages Erase: Address of the page which returned an error Program: Address which was selected for data program</li> </ul>
Return values	<ul style="list-style-type: none"> <li>none</li> </ul>

**16.2.11 HAL\_FLASH\_Unlock**

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Unlock (void )</b>
Function Description	Unlock the FLASH control register access.
Return values	<ul style="list-style-type: none"> <li>HAL Status</li> </ul>

**16.2.12 HAL\_FLASH\_Lock**

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Lock (void )</b>
Function Description	Locks the FLASH control register access.
Return values	<ul style="list-style-type: none"> <li>HAL Status</li> </ul>

### 16.2.13 HAL\_FLASH\_OB\_Unlock

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void )</b>
Function Description	Unlock the FLASH Option Control Registers access.
Return values	<ul style="list-style-type: none"><li>• HAL Status</li></ul>

### 16.2.14 HAL\_FLASH\_OB\_Lock

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Lock (void )</b>
Function Description	Lock the FLASH Option Control Registers access.
Return values	<ul style="list-style-type: none"><li>• HAL Status</li></ul>

### 16.2.15 HAL\_FLASH\_OB\_Launch

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Launch (void )</b>
Function Description	Launch the option byte loading.
Return values	<ul style="list-style-type: none"><li>• HAL_StatusTypeDef HAL Status</li></ul>
Notes	<ul style="list-style-type: none"><li>• This function will reset automatically the MCU.</li></ul>

### 16.2.16 HAL\_FLASH\_GetError

Function Name	<b>uint32_t HAL_FLASH_GetError (void )</b>
Function Description	Get the specific FLASH error flag.
Return values	<ul style="list-style-type: none"><li>• FLASH_ErrorCode The returned value can be: FLASH_ERROR_PG: FLASH Programming error flag FLASH_ERROR_WRP: FLASH Write protected error flag</li></ul>

## 16.3 FLASH Firmware driver defines

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

### 16.3.1 FLASH

FLASH

#### **FLASH Error Codes**

FLASH\_ERROR\_NONE

FLASH\_ERROR\_PG

FLASH\_ERROR\_WRP

FLASH\_ERROR\_OPTV

#### **FLASH Exported Macros**

<code>__HAL_FLASH_HALF_CYCLE_ACCESS_ENABLE</code>	<b>Description:</b> <ul style="list-style-type: none"> <li>Enable the FLASH half cycle access.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None:</li> </ul>
<code>__HAL_FLASH_HALF_CYCLE_ACCESS_DISABLE</code>	<b>Description:</b> <ul style="list-style-type: none"> <li>Disable the FLASH half cycle access.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None:</li> </ul>

**Flag definition**

<code>FLASH_FLAG_BSY</code>	FLASH Bank1 Busy flag
<code>FLASH_FLAG_PGERR</code>	FLASH Bank1 Programming error flag
<code>FLASH_FLAG_WRPERR</code>	FLASH Bank1 Write protected error flag
<code>FLASH_FLAG_EOP</code>	FLASH Bank1 End of Operation flag
<code>FLASH_FLAG_BSY_BANK1</code>	FLASH Bank1 Busy flag
<code>FLASH_FLAG_PGERR_BANK1</code>	FLASH Bank1 Programming error flag
<code>FLASH_FLAG_WRPERR_BANK1</code>	FLASH Bank1 Write protected error flag
<code>FLASH_FLAG_EOP_BANK1</code>	FLASH Bank1 End of Operation flag
<code>FLASH_FLAG_BSY_BANK2</code>	FLASH Bank2 Busy flag
<code>FLASH_FLAG_PGERR_BANK2</code>	FLASH Bank2 Programming error flag
<code>FLASH_FLAG_WRPERR_BANK2</code>	FLASH Bank2 Write protected error flag
<code>FLASH_FLAG_EOP_BANK2</code>	FLASH Bank2 End of Operation flag
<code>FLASH_FLAG_OPTVERR</code>	Option Byte Error

**Interrupt**

<code>__HAL_FLASH_ENABLE_IT</code>	<b>Description:</b> <ul style="list-style-type: none"> <li>Enable the specified FLASH interrupt.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><code>__INTERERRUPT__</code>: FLASH interrupt This parameter can be any combination of the following values: <ul style="list-style-type: none"> <li><code>FLASH_IT_EOP_BANK1</code>: End of FLASH Operation Interrupt on bank1</li> <li><code>FLASH_IT_ERR_BANK1</code>: Error Interrupt on bank1</li> <li><code>FLASH_IT_EOP_BANK2</code>: End of FLASH Operation Interrupt on bank2</li> <li><code>FLASH_IT_ERR_BANK2</code>: Error Interrupt on bank2</li> </ul> </li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>none:</li> </ul>
------------------------------------	--

**\_\_HAL\_FLASH\_DISABLE\_IT****Description:**

- Disable the specified FLASH interrupt.

**Parameters:**

- **\_\_INTERRUPT\_\_**: FLASH interrupt This parameter can be any combination of the following values:
  - FLASH\_IT\_EOP\_BANK1: End of FLASH Operation Interrupt on bank1
  - FLASH\_IT\_ERR\_BANK1: Error Interrupt on bank1
  - FLASH\_IT\_EOP\_BANK2: End of FLASH Operation Interrupt on bank2
  - FLASH\_IT\_ERR\_BANK2: Error Interrupt on bank2

**Return value:**

- none:

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

- Get the specified FLASH flag status.

**Parameters:**

- **\_\_FLAG\_\_**: specifies the FLASH flag to check. This parameter can be one of the following values:
  - FLASH\_FLAG\_EOP\_BANK1 : FLASH End of Operation flag on bank1
  - FLASH\_FLAG\_WRPERR\_BANK1: FLASH Write protected error flag on bank1
  - FLASH\_FLAG\_PGERR\_BANK1 : FLASH Programming error flag on bank1
  - FLASH\_FLAG\_BSY\_BANK1 : FLASH Busy flag on bank1
  - FLASH\_FLAG\_EOP\_BANK2 : FLASH End of Operation flag on bank2
  - FLASH\_FLAG\_WRPERR\_BANK2: FLASH Write protected error flag on bank2
  - FLASH\_FLAG\_PGERR\_BANK2 : FLASH Programming error flag on bank2
  - FLASH\_FLAG\_BSY\_BANK2 : FLASH Busy flag on bank2
  - FLASH\_FLAG\_OPTVERR : Loaded OB and its complement do not match

**Return value:**

- The: new state of **\_\_FLAG\_\_** (SET or RESET).

**\_\_HAL\_FLASH\_CLEAR\_FLAG****Description:**

- Clear the specified FLASH flag.

**Parameters:**

- **\_\_FLAG\_\_**: specifies the FLASH flags to clear. This parameter can be any combination of the following values:

- FLASH\_FLAG\_EOP\_BANK1 : FLASH End of Operation flag on bank1
- FLASH\_FLAG\_WRPERR\_BANK1: FLASH Write protected error flag on bank1
- FLASH\_FLAG\_PGERR\_BANK1 : FLASH Programming error flag on bank1
- FLASH\_FLAG\_BSY\_BANK1 : FLASH Busy flag on bank1
- FLASH\_FLAG\_EOP\_BANK2 : FLASH End of Operation flag on bank2
- FLASH\_FLAG\_WRPERR\_BANK2: FLASH Write protected error flag on bank2
- FLASH\_FLAG\_PGERR\_BANK2 : FLASH Programming error flag on bank2
- FLASH\_FLAG\_BSY\_BANK2 : FLASH Busy flag on bank2
- FLASH\_FLAG\_OPTVERR : Loaded OB and its complement do not match

**Return value:**

- none:

**Interrupt definition**

FLASH_IT_EOP	End of FLASH Operation Interrupt source Bank1
FLASH_IT_ERR	Error Interrupt source Bank1
FLASH_IT_EOP_BANK1	End of FLASH Operation Interrupt source Bank1
FLASH_IT_ERR_BANK1	Error Interrupt source Bank1
FLASH_IT_EOP_BANK2	End of FLASH Operation Interrupt source Bank2
FLASH_IT_ERR_BANK2	Error Interrupt source Bank2

**Latency configuration**

`__HAL_FLASH_SET_LATENCY` **Description:**

- Set the FLASH Latency.

**Parameters:**

- `__LATENCY__`: FLASH Latency This parameter can be one of the following values:
  - `FLASH_LATENCY_0`: FLASH Zero Latency cycle
  - `FLASH_LATENCY_1`: FLASH One Latency cycle
  - `FLASH_LATENCY_2`: FLASH Two Latency cycle

**Return value:**

- None:

`__HAL_FLASH_GET_LATENCY` **Description:**

- Get the FLASH Latency.

**Return value:**

- FLASH: Latency This parameter can be one of the following values:
  - FLASH\_LATENCY\_0: FLASH Zero Latency cycle
  - FLASH\_LATENCY\_1: FLASH One Latency cycle
  - FLASH\_LATENCY\_2: FLASH Two Latency cycle

**Latency Values**

FLASH\_LATENCY\_0 FLASH Zero Latency cycle

FLASH\_LATENCY\_1 FLASH One Latency cycle

FLASH\_LATENCY\_2 FLASH Two Latency cycles

**Prefetch activation or deactivation**

\_\_HAL\_FLASH\_PREFETCH\_BUFFER\_ENABLE

**Description:**

- Enable the FLASH prefetch buffer.

**Return value:**

- None:

\_\_HAL\_FLASH\_PREFETCH\_BUFFER\_DISABLE

**Description:**

- Disable the FLASH prefetch buffer.

**Return value:**

- None:

**FLASH Private Constants**

FLASH\_TIMEOUT\_VALUE

**FLASH Private Macros**

IS\_FLASH\_TYPEPROGRAM

**Type Program**

FLASH\_TYPEPROGRAM\_HALFWORD

Program a half-word (16-bit) at a specified address.

FLASH\_TYPEPROGRAM\_WORD

Program a word (32-bit) at a specified address.

FLASH\_TYPEPROGRAM\_DOUBLEWORD

Program a double word (64-bit) at a specified address



## 17 HAL FLASH Extension Driver

### 17.1 FLASHEx Firmware driver registers structures

#### 17.1.1 FLASH\_EraseInitTypeDef

*FLASH\_EraseInitTypeDef* is defined in the stm32f1xx\_hal\_flash\_ex.h

##### Data Fields

- *uint32\_t TypeErase*
- *uint32\_t Banks*
- *uint32\_t PageAddress*
- *uint32\_t NbPages*

##### Field Documentation

- *uint32\_t FLASH\_EraseInitTypeDef::TypeErase* TypeErase: Mass erase or page erase. This parameter can be a value of [FLASHEx\\_Type\\_Erase](#)
- *uint32\_t FLASH\_EraseInitTypeDef::Banks* Select banks to erase when Mass erase is enabled. This parameter must be a value of [FLASHEx\\_Banks](#)
- *uint32\_t FLASH\_EraseInitTypeDef::PageAddress* PageAddress: Initial FLASH page address to erase when mass erase is disabled This parameter must be a number between Min\_Data = 0x08000000 and Max\_Data = FLASH\_BANKx\_END (x = 1 or 2 depending on devices)
- *uint32\_t FLASH\_EraseInitTypeDef::NbPages* NbPages: Number of pages to be erased. This parameter must be a value between Min\_Data = 1 and Max\_Data = (max number of pages - value of initial page)

#### 17.1.2 FLASH\_OBProgramInitTypeDef

*FLASH\_OBProgramInitTypeDef* is defined in the stm32f1xx\_hal\_flash\_ex.h

##### Data Fields

- *uint32\_t OptionType*
- *uint32\_t WRPState*
- *uint32\_t WRPPage*
- *uint32\_t Banks*
- *uint8\_t RDPLLevel*
- *uint8\_t USERConfig*
- *uint32\_t DATAAddress*
- *uint8\_t DATAData*

##### Field Documentation

- *uint32\_t FLASH\_OBProgramInitTypeDef::OptionType* OptionType: Option byte to be configured. This parameter can be a value of [FLASHEx\\_OB\\_Type](#)
- *uint32\_t FLASH\_OBProgramInitTypeDef::WRPState* WRPState: Write protection activation or deactivation. This parameter can be a value of [FLASHEx\\_OB\\_WRP\\_State](#)

- **`uint32_t FLASH_OBProgramInitTypeDef::WRPPage`** WRPPage: specifies the page(s) to be write protected This parameter can be a value of [FLASHEx\\_OB\\_Write\\_Protection](#)
- **`uint32_t FLASH_OBProgramInitTypeDef::Banks`** Select banks for WRP activation/deactivation of all sectors. This parameter must be a value of [FLASHEx\\_Banks](#)
- **`uint8_t FLASH_OBProgramInitTypeDef::RDPLLevel`** RDPLLevel: Set the read protection level.. This parameter can be a value of [FLASHEx\\_OB\\_Read\\_Protection](#)
- **`uint8_t FLASH_OBProgramInitTypeDef::USERConfig`** USERConfig: Program the FLASH User Option Byte: IWDG / STOP / STDBY / BOOT1 This parameter can be a combination of [FLASHEx\\_OB\\_IWatchdog](#), [FLASHEx\\_OB\\_nRST\\_STOP](#), [FLASHEx\\_OB\\_nRST\\_STDBY](#), [FLASHEx\\_OB\\_BOOT1](#)
- **`uint32_t FLASH_OBProgramInitTypeDef::DATAAddress`** DATAAddress: Address of the option byte DATA to be programmed This parameter can be a value of [FLASHEx\\_OB\\_Data\\_Address](#)
- **`uint8_t FLASH_OBProgramInitTypeDef::DATADData`** DATADData: Data to be stored in the option byte DATA This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF

## 17.2 FLASHEx Firmware driver API description

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

### 17.2.1 IO operation functions

- [HAL\\_FLASHEx\\_Erase\(\)](#)
- [HAL\\_FLASHEx\\_Erase\\_IT\(\)](#)

### 17.2.2 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

- [HAL\\_FLASHEx\\_OBerase\(\)](#)
- [HAL\\_FLASHEx\\_OBProgram\(\)](#)
- [HAL\\_FLASHEx\\_OBGetConfig\(\)](#)

### 17.2.3 HAL\_FLASHEx\_Erase

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_Erase</b> (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)
Function Description	Perform a mass erase or erase the specified FLASH memory pages.
Parameters	<ul style="list-style-type: none"> <li>• <b>pEraseInit</b>: pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.</li> <li>• <b>PageError</b>: pointer to variable that contains the configuration information on faulty page in case of error (0xFFFFFFFF means that all the pages have been correctly erased)</li> </ul>

Return values	<ul style="list-style-type: none"> <li>• HAL_StatusTypeDef HAL Status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interface</li> </ul>

### 17.2.4 HAL\_FLASHEx\_Erase\_IT

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)</b>
Function Description	Perform a mass erase or erase the specified FLASH memory sectors with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>pEraseInit</b>: pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL_StatusTypeDef HAL Status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interface</li> </ul>

### 17.2.5 HAL\_FLASHEx\_OB\_Erase

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_OB_Erase (void )</b>
Function Description	Erases the FLASH option bytes.
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This functions erases all option bytes except the Read protection (RDP). The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_OB_Unlock() should be called before to unlock the options bytes The function HAL_FLASH_OB_Launch() should be called after to force the reload of the options bytes (system reset will occur)</li> </ul>

### 17.2.6 HAL\_FLASHEx\_OBProgram

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)</b>
Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none"> <li>• <b>pOBInit</b>: pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL_StatusTypeDef HAL Status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_OB_Unlock() should be called before to unlock</li> </ul>

the options bytes The function HAL\_FLASH\_OB\_Launch() should be called after to force the reload of the options bytes (system reset will occur)

### 17.2.7 HAL\_FLASHEx\_OBGetConfig

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

## 17.3 FLASHEx Firmware driver defines

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

### 17.3.1 FLASHEx

FLASHEx

#### **Banks**

FLASH_BANK_1	Bank 1
FLASH_BANK_2	Bank 2
FLASH_BANK_BOTH	Bank1 and Bank2

#### **Option Byte BOOT1**

OB_BOOT1_RESET	BOOT1 Reset
OB_BOOT1_SET	BOOT1 Set

#### **Option Byte Data Address**

OB_DATA_ADDRESS_DATA0	
OB_DATA_ADDRESS_DATA1	

#### **Option Byte IWatchdog**

OB_IWDG_SW	Software IWDG selected
OB_IWDG_HW	Hardware IWDG selected

#### **Option Byte nRST STDBY**

OB_STDBY_NO_RST	No reset generated when entering in STANDBY
OB_STDBY_RST	Reset generated when entering in STANDBY

#### **Option Byte nRST STOP**

OB_STOP_NO_RST	No reset generated when entering in STOP
OB_STOP_RST	Reset generated when entering in STOP

#### **Option Byte Read Protection**

OB_RDP_LEVEL_0	
----------------	--

OB\_RDP\_LEVEL\_1

**Option Bytes Type**

OPTIONBYTE\_WRP    WRP option byte configuration

OPTIONBYTE\_RDP    RDP option byte configuration

OPTIONBYTE\_USER    USER option byte configuration

OPTIONBYTE\_DATA    DATA option byte configuration

**Option Bytes Write Protection**

OB\_WRP\_PAGES0TO1            Write protection of page 0 TO 1

OB\_WRP\_PAGES2TO3            Write protection of page 2 TO 3

OB\_WRP\_PAGES4TO5            Write protection of page 4 TO 5

OB\_WRP\_PAGES6TO7            Write protection of page 6 TO 7

OB\_WRP\_PAGES8TO9            Write protection of page 8 TO 9

OB\_WRP\_PAGES10TO11            Write protection of page 10 TO 11

OB\_WRP\_PAGES12TO13            Write protection of page 12 TO 13

OB\_WRP\_PAGES14TO15            Write protection of page 14 TO 15

OB\_WRP\_PAGES16TO17            Write protection of page 16 TO 17

OB\_WRP\_PAGES18TO19            Write protection of page 18 TO 19

OB\_WRP\_PAGES20TO21            Write protection of page 20 TO 21

OB\_WRP\_PAGES22TO23            Write protection of page 22 TO 23

OB\_WRP\_PAGES24TO25            Write protection of page 24 TO 25

OB\_WRP\_PAGES26TO27            Write protection of page 26 TO 27

OB\_WRP\_PAGES28TO29            Write protection of page 28 TO 29

OB\_WRP\_PAGES30TO31            Write protection of page 30 TO 31

OB\_WRP\_PAGES32TO33            Write protection of page 32 TO 33

OB\_WRP\_PAGES34TO35            Write protection of page 34 TO 35

OB\_WRP\_PAGES36TO37            Write protection of page 36 TO 37

OB\_WRP\_PAGES38TO39            Write protection of page 38 TO 39

OB\_WRP\_PAGES40TO41            Write protection of page 40 TO 41

OB\_WRP\_PAGES42TO43            Write protection of page 42 TO 43

OB\_WRP\_PAGES44TO45            Write protection of page 44 TO 45

OB\_WRP\_PAGES46TO47            Write protection of page 46 TO 47

OB\_WRP\_PAGES48TO49            Write protection of page 48 TO 49

OB\_WRP\_PAGES50TO51            Write protection of page 50 TO 51

OB\_WRP\_PAGES52TO53            Write protection of page 52 TO 53

OB\_WRP\_PAGES54TO55            Write protection of page 54 TO 55

OB\_WRP\_PAGES56TO57            Write protection of page 56 TO 57

---

OB_WRP_PAGES58TO59	Write protection of page 58 TO 59
OB_WRP_PAGES60TO61	Write protection of page 60 TO 61
OB_WRP_PAGES62TO127	Write protection of page 62 TO 127
OB_WRP_PAGES62TO255	Write protection of page 62 TO 255
OB_WRP_PAGES62TO511	Write protection of page 62 TO 511
OB_WRP_ALLPAGES	Write protection of all Pages
OB_WRP_PAGES0TO15MASK	
OB_WRP_PAGES16TO31MASK	
OB_WRP_PAGES32TO47MASK	
OB_WRP_PAGES48TO511MASK	

**Option Byte WRP State**

OB_WRPSTATE_DISABLE	Disable the write protection of the desired pages
OB_WRPSTATE_ENABLE	Enable the write protection of the desired pages

**FLASHEx Private Constants**

FLASH\_SIZE\_DATA\_REGISTER  
OBR\_REG\_INDEX  
SR\_FLAG\_MASK  
FLASH\_PAGE\_SIZE

**FLASHEx Private Macros**

IS\_FLASH\_TYPEERASE  
IS\_OPTIONBYTE  
IS\_WRPSTATE  
IS\_OB\_RDP\_LEVEL  
IS\_OB\_DATA\_ADDRESS  
IS\_OB\_IWDG\_SOURCE  
IS\_OB\_STOP\_SOURCE  
IS\_OB\_STDBY\_SOURCE  
IS\_OB\_BOOT1  
IS\_FLASH\_NB\_PAGES  
IS\_OB\_WRP  
IS\_FLASH\_BANK  
IS\_FLASH\_PROGRAM\_ADDRESS  
IS\_FLASH\_LATENCY

**Type Erase**

FLASH_TYPEERASE_PAGES	Pages erase only
FLASH_TYPEERASE_MASSERASE	Flash mass erase activation

## 18 HAL GPIO Generic Driver

### 18.1 GPIO Firmware driver registers structures

#### 18.1.1 GPIO\_InitTypeDef

*GPIO\_InitTypeDef* is defined in the `stm32f1xx_hal_gpio.h`

##### Data Fields

- *uint32\_t Pin*
- *uint32\_t Mode*
- *uint32\_t Pull*
- *uint32\_t Speed*

##### 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](#)

### 18.2 GPIO Firmware driver API description

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

#### 18.2.1 GPIO Peripheral features

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the General Purpose IO (GPIO) Ports, can be individually configured by software in several modes:

- Input mode
- Analog mode
- Output mode
- Alternate function mode
- External interrupt/event lines

During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.

All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.

In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.

All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.

The external interrupt/event controller consists of up to 20 edge detectors in connectivity line devices, or 19 edge detectors in other devices for generating event/interrupt requests. Each input line can be independently configured to select the type (event or interrupt) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently. A pending register maintains the status line of the interrupt requests.

## 18.2.2 How to use this driver

1. Enable the GPIO APB2 clock using the following function :  
\_\_HAL\_GPIOx\_CLK\_ENABLE().
2. Configure the GPIO pin(s) using HAL\_GPIO\_Init().
  - Configure the IO mode using "Mode" member from GPIO\_InitTypeDef structure
  - Activate Pull-up, Pull-down resistor using "Pull" member from GPIO\_InitTypeDef structure.
  - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from GPIO\_InitTypeDef structure
  - Analog mode is required when a pin is to be used as ADC channel or DAC output.
  - In case of external interrupt/event selection the "Mode" member from GPIO\_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL\_NVIC\_SetPriority() and enable it using HAL\_NVIC\_EnableIRQ().
4. To get the level of a pin configured in input mode use HAL\_GPIO\_ReadPin().
5. To set/reset the level of a pin configured in output mode use HAL\_GPIO\_WritePin()/HAL\_GPIO\_TogglePin().
6. To lock pin configuration until next reset use HAL\_GPIO\_LockPin().
7. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
8. The LSE oscillator pins OSC32\_IN and OSC32\_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
9. The HSE oscillator pins OSC\_IN/OSC\_OUT can be used as general purpose PD0 and PD1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

## 18.2.3 Initialization and deinitialization functions

This section provides functions allowing to initialize and de-initialize the GPIOs to be ready for use.

- [\*\*HAL\\_GPIO\\_Init\(\)\*\*](#)



- [HAL\\_GPIO\\_DeInit\(\)](#)

## 18.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the GPIOs.

- [HAL\\_GPIO\\_ReadPin\(\)](#)
- [HAL\\_GPIO\\_WritePin\(\)](#)
- [HAL\\_GPIO\\_TogglePin\(\)](#)
- [HAL\\_GPIO\\_LockPin\(\)](#)
- [HAL\\_GPIO\\_EXTI\\_IRQHandler\(\)](#)
- [HAL\\_GPIO\\_EXTI\\_Callback\(\)](#)

## 18.2.5 HAL\_GPIO\_Init

Function Name	<b>void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)</b>
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b>: where x can be (A..G depending on device used) to select the GPIO peripheral</li> <li>• <b>GPIO_Init</b>: pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 18.2.6 HAL\_GPIO\_DeInit

Function Name	<b>void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)</b>
Function Description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b>: where x can be (A..G depending on device used) to select the GPIO peripheral</li> <li>• <b>GPIO_Pin</b>: specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 18.2.7 HAL\_GPIO\_ReadPin

Function Name	<b>GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</b>
Function Description	Reads the specified input port pin.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b>: where x can be (A..G depending on device used) to select the GPIO peripheral</li> </ul>

- **GPIO\_Pin:** specifies the port bit to read. This parameter can be GPIO\_PIN\_x where x can be (0..15).
- Return values
- The input port pin value.

### 18.2.8 HAL\_GPIO\_WritePin

- Function Name      **void HAL\_GPIO\_WritePin (GPIO\_TypeDef \* GPIOx, uint16\_t GPIO\_Pin, GPIO\_PinState PinState)**
- Function Description      Sets or clears the selected data port bit.
- Parameters
- **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral
  - **GPIO\_Pin:** specifies the port bit to be written. This parameter can be one of GPIO\_PIN\_x where x can be (0..15).
  - **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO\_PinState enum values: GPIO\_BIT\_RESET: to clear the port pin  
GPIO\_BIT\_SET: to set the port pin
- Return values
- None
- Notes
- This function uses GPIOx\_BSRR register to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

### 18.2.9 HAL\_GPIO\_TogglePin

- Function Name      **void HAL\_GPIO\_TogglePin (GPIO\_TypeDef \* GPIOx, uint16\_t GPIO\_Pin)**
- Function Description      Toggles the specified GPIO pin.
- Parameters
- **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral
  - **GPIO\_Pin:** Specifies the pins to be toggled.
- Return values
- None

### 18.2.10 HAL\_GPIO\_LockPin

- Function Name      **HAL\_StatusTypeDef HAL\_GPIO\_LockPin (GPIO\_TypeDef \* GPIOx, uint16\_t GPIO\_Pin)**
- Function Description      Locks GPIO Pins configuration registers.
- Parameters
- **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral
  - **GPIO\_Pin:** specifies the port bit to be locked. This parameter can be any combination of GPIO\_Pin\_x where x can be (0..15).
- Return values
- None

## Notes

- The locking mechanism allows the IO configuration to be frozen. When the LOCK sequence has been applied on a port bit, it is no longer possible to modify the value of the port bit until the next reset.

### 18.2.11 HAL\_GPIO\_EXTI\_IRQHandler

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

### 18.2.12 HAL\_GPIO\_EXTI\_Callback

Function Name	<b>void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)</b>
Function Description	EXTI line detection callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIO_Pin:</b> Specifies the pins connected EXTI line</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 18.3 GPIO Firmware driver defines

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

### 18.3.1 GPIO

GPIO

#### *GPIO Exported Macros*

`__HAL_GPIO_EXTI_GET_FLAG`

#### **Description:**

- Checks whether the specified EXTI line flag is set or not.

#### **Parameters:**

- `__EXTI_LINE__`: specifies the EXTI line flag to check. This parameter can be `GPIO_PIN_x` where x can be (0..15)

#### **Return value:**

- The: new state of `__EXTI_LINE__` (SET or RESET).

`__HAL_GPIO_EXTI_CLEAR_FLAG`

#### **Description:**

- Clears the EXTI's line pending flags.

#### **Parameters:**

- `__EXTI_LINE__`: specifies the EXTI lines flags to clear. This parameter can be any combination of `GPIO_PIN_x` where x can

be (0..15)

**Return value:**

- None:

**Description:**

- Checks whether the specified EXTI line is asserted or not.

**Parameters:**

- `__EXTI_LINE__`: specifies the EXTI line to check. This parameter can be `GPIO_PIN_x` where x can be (0..15)

**Return value:**

- The: new state of `__EXTI_LINE__` (SET or RESET).

**Description:**

- Clears the EXTI's line pending bits.

**Parameters:**

- `__EXTI_LINE__`: specifies the EXTI lines to clear. This parameter can be any combination of `GPIO_PIN_x` where x can be (0..15)

**Return value:**

- None:

**Description:**

- Generates a Software interrupt on selected EXTI line.

**Parameters:**

- `__EXTI_LINE__`: specifies the EXTI line to check. This parameter can be `GPIO_PIN_x` where x can be (0..15)

**Return value:**

- None:

`__HAL_GPIO_EXTI_GET_IT`

`__HAL_GPIO_EXTI_CLEAR_IT`

`__HAL_GPIO_EXTI_GENERATE_SWIT`

**GPIO mode define**

`GPIO_MODE_INPUT`

Input Floating Mode

`GPIO_MODE_OUTPUT_PP`

Output Push Pull Mode

`GPIO_MODE_OUTPUT_OD`

Output Open Drain Mode

`GPIO_MODE_AF_PP`

Alternate Function Push Pull Mode

`GPIO_MODE_AF_OD`

Alternate Function Open Drain Mode

`GPIO_MODE_AF_INPUT`

Alternate Function Input Mode

`GPIO_MODE_ANALOG`

Analog Mode

---

GPIO_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
GPIO_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
GPIO_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
GPIO_MODE_EVT_RISING	External Event Mode with Rising edge trigger detection
GPIO_MODE_EVT_FALLING	External Event Mode with Falling edge trigger detection
GPIO_MODE_EVT_RISING_FALLING	External Event Mode with Rising/Falling edge trigger detection

***GPIO pins define***

GPIO\_PIN\_0  
GPIO\_PIN\_1  
GPIO\_PIN\_2  
GPIO\_PIN\_3  
GPIO\_PIN\_4  
GPIO\_PIN\_5  
GPIO\_PIN\_6  
GPIO\_PIN\_7  
GPIO\_PIN\_8  
GPIO\_PIN\_9  
GPIO\_PIN\_10  
GPIO\_PIN\_11  
GPIO\_PIN\_12  
GPIO\_PIN\_13  
GPIO\_PIN\_14  
GPIO\_PIN\_15  
GPIO\_PIN\_All  
GPIO\_PIN\_MASK

***GPIO Private Constants***

GPIO\_MODE  
EXTI\_MODE  
GPIO\_MODE\_IT  
GPIO\_MODE\_EVT  
RISING\_EDGE  
FALLING\_EDGE

GPIO\_OUTPUT\_TYPE

GPIO\_NUMBER

GPIO\_CR\_MODE\_INPUT                    00: Input mode (reset state)

GPIO\_CR\_CNF\_ANALOG                    00: Analog mode

GPIO\_CR\_CNF\_INPUT\_FLOATING           01: Floating input (reset state)

GPIO\_CR\_CNF\_INPUT\_PU\_PD              10: Input with pull-up / pull-down

GPIO\_CR\_CNF\_GP\_OUTPUT\_PP             00: General purpose output push-pull

GPIO\_CR\_CNF\_GP\_OUTPUT\_OD             01: General purpose output Open-drain

GPIO\_CR\_CNF\_AF\_OUTPUT\_PP             10: Alternate function output Push-pull

GPIO\_CR\_CNF\_AF\_OUTPUT\_OD             11: Alternate function output Open-drain

**GPIO\_Private\_Macros**

IS\_GPIO\_PIN\_ACTION

IS\_GPIO\_PIN

IS\_GPIO\_PULL

IS\_GPIO\_SPEED

IS\_GPIO\_MODE

**GPIO pull define**

GPIO\_NOPULL                    No Pull-up or Pull-down activation

GPIO\_PULLUP                    Pull-up activation

GPIO\_PULLDOWN                Pull-down activation

**GPIO speed define**

GPIO\_SPEED\_LOW                Low speed

GPIO\_SPEED\_MEDIUM            Medium speed

GPIO\_SPEED\_HIGH               High speed

## 19 HAL GPIO Extension Driver

### 19.1 GPIOEx Firmware driver API description

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

#### 19.1.1 GPIO Peripheral extension features

GPIO module on STM32F1 family, manage also the AFIO register:

- Possibility to use the EVENTOUT Cortex feature

#### 19.1.2 How to use this driver

This driver provides functions to use EVENTOUT Cortex feature

1. Configure EVENTOUT Cortex feature using the function `HAL_GPIOEx_ConfigEventout()`
2. Activate EVENTOUT Cortex feature using the `HAL_GPIOEx_EnableEventout()`
3. Deactivate EVENTOUT Cortex feature using the `HAL_GPIOEx_DisableEventout()`

#### 19.1.3 Extended features functions

This section provides functions allowing to:

- Configure EVENTOUT Cortex feature using the function `HAL_GPIOEx_ConfigEventout()`
- Activate EVENTOUT Cortex feature using the `HAL_GPIOEx_EnableEventout()`
- Deactivate EVENTOUT Cortex feature using the `HAL_GPIOEx_DisableEventout()`
- [`HAL\_GPIOEx\_ConfigEventout\(\)`](#)
- [`HAL\_GPIOEx\_EnableEventout\(\)`](#)
- [`HAL\_GPIOEx\_DisableEventout\(\)`](#)

#### 19.1.4 HAL\_GPIOEx\_ConfigEventout

Function Name	<b><code>void HAL_GPIOEx_ConfigEventout (uint32_t GPIO_PortSource, uint32_t GPIO_PinSource)</code></b>
Function Description	Configures the port and pin on which the EVENTOUT Cortex signal will be connected.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIO_PortSource:</b> Select the port used to output the Cortex EVENTOUT signal. This parameter can be a value of EVENTOUT Port.</li> <li>• <b>GPIO_PinSource:</b> Select the pin used to output the Cortex EVENTOUT signal. This parameter can be a value of EVENTOUT Pin.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 19.1.5 HAL\_GPIOEx\_EnableEventout

Function Name	<b>void HAL_GPIOEx_EnableEventout (void )</b>
Function Description	Enables the Event Output.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 19.1.6 HAL\_GPIOEx\_DisableEventout

Function Name	<b>void HAL_GPIOEx_DisableEventout (void )</b>
Function Description	Disables the Event Output.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 19.2 GPIOEx Firmware driver defines

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

### 19.2.1 GPIOEx

GPIOEx

#### *Alternate Function Remapping*

`__HAL_AFIO_REMAP_SPI1_ENABLE`

#### **Description:**

- Enable the remapping of SPI1 alternate function NSS, SCK, MISO and MOSI.

#### **Return value:**

- None:

`__HAL_AFIO_REMAP_SPI1_DISABLE`

#### **Description:**

- Disable the remapping of SPI1 alternate function NSS, SCK, MISO and MOSI.

#### **Return value:**

- None:

`__HAL_AFIO_REMAP_I2C1_ENABLE`

#### **Description:**

- Enable the remapping of I2C1 alternate function SCL and SDA.

#### **Return value:**

- None:

`__HAL_AFIO_REMAP_I2C1_DISABLE`

#### **Description:**

- Disable the remapping of I2C1 alternate function SCL and SDA.



\_\_HAL\_AFIO\_REMAP\_USART1\_ENABLE

**Return value:**

- None:

**Description:**

- Enable the remapping of USART1 alternate function TX and RX.

**Return value:**

- None:

**Description:**

- Disable the remapping of USART1 alternate function TX and RX.

\_\_HAL\_AFIO\_REMAP\_USART1\_DISABLE

**Return value:**

- None:

**Description:**

- Enable the remapping of USART2 alternate function CTS, RTS, CK, TX and RX.

\_\_HAL\_AFIO\_REMAP\_USART2\_ENABLE

**Return value:**

- None:

**Description:**

- Disable the remapping of USART2 alternate function CTS, RTS, CK, TX and RX.

\_\_HAL\_AFIO\_REMAP\_USART2\_DISABLE

**Return value:**

- None:

**Description:**

- Enable the remapping of USART3 alternate function CTS, RTS, CK, TX and RX.

\_\_HAL\_AFIO\_REMAP\_USART3\_ENABLE

**Return value:**

- None:

**Description:**

- Enable the remapping of USART3 alternate function CTS, RTS, CK, TX and RX.

\_\_HAL\_AFIO\_REMAP\_USART3\_PARTIAL

**Return value:**

- None:

**Description:**

- Disable the remapping of USART3 alternate function

\_\_HAL\_AFIO\_REMAP\_USART3\_DISABLE

CTS, RTS, CK, TX and RX.

**Return value:**

- None:

**Description:**

- Enable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)

**Return value:**

- None:

**Description:**

- Enable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)

**Return value:**

- None:

**Description:**

- Disable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)

**Return value:**

- None:

**Description:**

- Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

**Return value:**

- None:

**Description:**

- Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

**Return value:**

- None:

**Description:**

- Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

\_\_HAL\_AFIO\_REMAP\_TIM1\_ENABLE

\_\_HAL\_AFIO\_REMAP\_TIM1\_PARTIAL

\_\_HAL\_AFIO\_REMAP\_TIM1\_DISABLE

\_\_HAL\_AFIO\_REMAP\_TIM2\_ENABLE

\_\_HAL\_AFIO\_REMAP\_TIM2\_PARTIAL\_2

\_\_HAL\_AFIO\_REMAP\_TIM2\_PARTIAL\_1

\_\_HAL\_AFIO\_REMAP\_TIM2\_DISABLE

**Return value:**

- None:

**Description:**

- Disable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

**Return value:**

- None:

**Description:**

- Enable the remapping of TIM3 alternate function channels 1 to 4.

**Return value:**

- None:

**Description:**

- Enable the remapping of TIM3 alternate function channels 1 to 4.

**Return value:**

- None:

**Description:**

- Disable the remapping of TIM3 alternate function channels 1 to 4.

**Return value:**

- None:

**Description:**

- Enable the remapping of TIM4 alternate function channels 1 to 4.

**Return value:**

- None:

**Description:**

- Disable the remapping of TIM4 alternate function channels 1 to 4.

**Return value:**

- None:

**Description:**

- Enable or disable the

\_\_HAL\_AFIO\_REMAP\_TIM3\_ENABLE

\_\_HAL\_AFIO\_REMAP\_TIM3\_PARTIAL

\_\_HAL\_AFIO\_REMAP\_TIM3\_DISABLE

\_\_HAL\_AFIO\_REMAP\_TIM4\_ENABLE

\_\_HAL\_AFIO\_REMAP\_TIM4\_DISABLE

\_\_HAL\_AFIO\_REMAP\_CAN1\_1

remapping of CAN alternate function CAN\_RX and CAN\_TX in devices with a single CAN interface.

**Return value:**

- None:

**Description:**

- Enable or disable the remapping of CAN alternate function CAN\_RX and CAN\_TX in devices with a single CAN interface.

**Return value:**

- None:

**Description:**

- Enable or disable the remapping of CAN alternate function CAN\_RX and CAN\_TX in devices with a single CAN interface.

**Return value:**

- None:

**Description:**

- Enable the remapping of PD0 and PD1.

**Return value:**

- None:

**Description:**

- Disable the remapping of PD0 and PD1.

**Return value:**

- None:

**Description:**

- Enable the remapping of TIM5CH4.

**Return value:**

- None:

**Description:**

- Disable the remapping of TIM5CH4.

**Return value:**

\_\_HAL\_AFIO\_REMAP\_CAN1\_2

\_\_HAL\_AFIO\_REMAP\_CAN1\_3

\_\_HAL\_AFIO\_REMAP\_PD01\_ENABLE

\_\_HAL\_AFIO\_REMAP\_PD01\_DISABLE

\_\_HAL\_AFIO\_REMAP\_TIM5CH4\_ENABLE

\_\_HAL\_AFIO\_REMAP\_TIM5CH4\_DISABLE

\_\_HAL\_AFIO\_REMAP\_ADC1\_ETRGINJ\_ENABLE

- None:

**Description:**

- Enable the remapping of ADC1\_ETRGINJ (ADC 1 External trigger injected conversion).

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_ADC1\_ETRGINJ\_DISABLE

**Description:**

- Disable the remapping of ADC1\_ETRGINJ (ADC 1 External trigger injected conversion).

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_ADC1\_ETRGREG\_ENABLE

**Description:**

- Enable the remapping of ADC1\_ETRGREG (ADC 1 External trigger regular conversion).

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_ADC1\_ETRGREG\_DISABLE

**Description:**

- Disable the remapping of ADC1\_ETRGREG (ADC 1 External trigger regular conversion).

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_ADC2\_ETRGINJ\_ENABLE

**Description:**

- Enable the remapping of ADC2\_ETRGREG (ADC 2 External trigger injected conversion).

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_ADC2\_ETRGINJ\_DISABLE

**Description:**

- Disable the remapping of ADC2\_ETRGREG (ADC 2 External trigger injected conversion).

**Return value:**

\_\_HAL\_AFIO\_REMAP\_ADC2\_ETRGREG\_ENABLE

- None:

**Description:**

- Enable the remapping of ADC2\_ETRGREG (ADC 2 External trigger regular conversion).

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_ADC2\_ETRGREG\_DISABLE

**Description:**

- Disable the remapping of ADC2\_ETRGREG (ADC 2 External trigger regular conversion).

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_SWJ\_ENABLE

**Description:**

- Enable the Serial wire JTAG configuration.

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_SWJ\_NONJTRST

**Description:**

- Enable the Serial wire JTAG configuration.

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_SWJ\_NOJTAG

**Description:**

- Enable the Serial wire JTAG configuration.

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_SWJ\_DISABLE

**Description:**

- Disable the Serial wire JTAG configuration.

**Return value:**

- None:

\_\_HAL\_AFIO\_REMAP\_TIM9\_ENABLE

**Description:**

- Enable the remapping of TIM9\_CH1 and TIM9\_CH2.

**Return value:**

- None:

---

`__HAL_AFIO_REMAP_TIM9_DISABLE`**Description:**

- Disable the remapping of TIM9\_CH1 and TIM9\_CH2.

**Return value:**

- None:

`__HAL_AFIO_REMAP_TIM10_ENABLE`**Description:**

- Enable the remapping of TIM10\_CH1.

**Return value:**

- None:

`__HAL_AFIO_REMAP_TIM10_DISABLE`**Description:**

- Disable the remapping of TIM10\_CH1.

**Return value:**

- None:

`__HAL_AFIO_REMAP_TIM11_ENABLE`**Description:**

- Enable the remapping of TIM11\_CH1.

**Return value:**

- None:

`__HAL_AFIO_REMAP_TIM11_DISABLE`**Description:**

- Disable the remapping of TIM11\_CH1.

**Return value:**

- None:

`__HAL_AFIO_REMAP_TIM13_ENABLE`**Description:**

- Enable the remapping of TIM13\_CH1.

**Return value:**

- None:

`__HAL_AFIO_REMAP_TIM13_DISABLE`**Description:**

- Disable the remapping of TIM13\_CH1.

**Return value:**

- None:

`__HAL_AFIO_REMAP_TIM14_ENABLE`**Description:**

- Enable the remapping of TIM14\_CH1.

\_\_HAL\_AFIO\_REMAP\_TIM14\_DISABLE

**Return value:**

- None:

**Description:**

- Disable the remapping of TIM14\_CH1.

**Return value:**

- None:

**Description:**

- Controls the use of the optional FSMC\_NADV signal.

**Return value:**

- None:

**Description:**

- Controls the use of the optional FSMC\_NADV signal.

**Return value:**

- None:

\_\_HAL\_AFIO\_FSMCNADV\_DISCONNECTED

\_\_HAL\_AFIO\_FSMCNADV\_CONNECTED

**EVENTOUT Pin**

AFIO_EVENTOUT_PIN_0	EVENTOUT on pin 0
AFIO_EVENTOUT_PIN_1	EVENTOUT on pin 1
AFIO_EVENTOUT_PIN_2	EVENTOUT on pin 2
AFIO_EVENTOUT_PIN_3	EVENTOUT on pin 3
AFIO_EVENTOUT_PIN_4	EVENTOUT on pin 4
AFIO_EVENTOUT_PIN_5	EVENTOUT on pin 5
AFIO_EVENTOUT_PIN_6	EVENTOUT on pin 6
AFIO_EVENTOUT_PIN_7	EVENTOUT on pin 7
AFIO_EVENTOUT_PIN_8	EVENTOUT on pin 8
AFIO_EVENTOUT_PIN_9	EVENTOUT on pin 9
AFIO_EVENTOUT_PIN_10	EVENTOUT on pin 10
AFIO_EVENTOUT_PIN_11	EVENTOUT on pin 11
AFIO_EVENTOUT_PIN_12	EVENTOUT on pin 12
AFIO_EVENTOUT_PIN_13	EVENTOUT on pin 13
AFIO_EVENTOUT_PIN_14	EVENTOUT on pin 14
AFIO_EVENTOUT_PIN_15	EVENTOUT on pin 15
IS_AFIO_EVENTOUT_PIN	

**EVENTOUT Port**

AFIO\_EVENTOUT\_PORT\_A EVENTOUT on port A



AFIO\_EVENTOUT\_PORT\_B    EVENTOUT on port B  
AFIO\_EVENTOUT\_PORT\_C    EVENTOUT on port C  
AFIO\_EVENTOUT\_PORT\_D    EVENTOUT on port D  
AFIO\_EVENTOUT\_PORT\_E    EVENTOUT on port E  
IS\_AFIO\_EVENTOUT\_PORT

***GPIOEx Private Macros***

GPIO\_GET\_INDEX

## 20 HAL HCD Generic Driver

### 20.1 HCD Firmware driver registers structures

#### 20.1.1 HCD\_HandleTypeDef

*HCD\_HandleTypeDef* is defined in the stm32f1xx\_hal\_hcd.h

##### Data Fields

- *HCD\_TypeDef \* Instance*
- *HCD\_InitTypeDef Init*
- *HCD\_HCTypeDef hc*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HCD\_StateTypeDef State*
- *void \* pData*

##### Field Documentation

- *HCD\_TypeDef\* HCD\_HandleTypeDef::Instance* Register base address
- *HCD\_InitTypeDef HCD\_HandleTypeDef::Init* HCD required parameters
- *HCD\_HCTypeDef HCD\_HandleTypeDef::hc[15]* Host channels parameters
- *HAL\_LockTypeDef HCD\_HandleTypeDef::Lock* HCD peripheral status
- *\_\_IO HCD\_StateTypeDef HCD\_HandleTypeDef::State* HCD communication state
- *void\* HCD\_HandleTypeDef::pData* Pointer Stack Handler

### 20.2 HCD Firmware driver API description

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

#### 20.2.1 How to use this driver

1. Declare a HCD\_HandleTypeDef handle structure, for example: HCD\_HandleTypeDef hhcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL\_HCD\_Init() API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the HAL\_HCD\_MspInit() API:
  - a. Enable the HCD/USB Low Level interface clock using the following macro
    - \_\_HAL\_RCC\_OTGFS\_CLK\_ENABLE()
  - b. Initialize the related GPIO clocks
  - c. Configure HCD pin-out
  - d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
  - a. hhcd.pData = phost;
6. Enable HCD transmission and reception:
  - a. HAL\_HCD\_Start();

## 20.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- [\*HAL\\_HCD\\_Init\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_Init\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_Halt\(\)\*](#)
- [\*HAL\\_HCD\\_DeInit\(\)\*](#)
- [\*HAL\\_HCD\\_MspInit\(\)\*](#)
- [\*HAL\\_HCD\\_MspDeInit\(\)\*](#)

## 20.2.3 IO operation functions

- [\*HAL\\_HCD\\_HC\\_SubmitRequest\(\)\*](#)
- [\*HAL\\_HCD\\_IRQHandler\(\)\*](#)
- [\*HAL\\_HCD\\_SOF\\_Callback\(\)\*](#)
- [\*HAL\\_HCD\\_Connect\\_Callback\(\)\*](#)
- [\*HAL\\_HCD\\_Disconnect\\_Callback\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_NotifyURBChange\\_Callback\(\)\*](#)

## 20.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the HCD data transfers.

- [\*HAL\\_HCD\\_Start\(\)\*](#)
- [\*HAL\\_HCD\\_Stop\(\)\*](#)
- [\*HAL\\_HCD\\_ResetPort\(\)\*](#)

## 20.2.5 Peripheral State functions

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

- [\*HAL\\_HCD\\_GetState\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_GetURBState\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_GetXferCount\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_GetState\(\)\*](#)
- [\*HAL\\_HCD\\_GetCurrentFrame\(\)\*](#)
- [\*HAL\\_HCD\\_GetCurrentSpeed\(\)\*](#)

## 20.2.6 HAL\_HCD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HCD_Init (HCD_HandleTypeDef * hhcd)</b>
Function Description	Initialize the host driver.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 20.2.7 HAL\_HCD\_HC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)</b>
Function Description	Initialize a host channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> <li>• <b>ch_num</b>: Channel number. This parameter can be a value from 1 to 15</li> <li>• <b>epnum</b>: Endpoint number. This parameter can be a value from 1 to 15</li> <li>• <b>dev_address</b>: : Current device address This parameter can be a value from 0 to 255</li> <li>• <b>speed</b>: Current device speed. This parameter can be one of these values: HCD_SPEED_FULL: Full speed mode, HCD_SPEED_LOW: Low speed mode</li> <li>• <b>ep_type</b>: Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type, EP_TYPE_ISOC: Isochronous type, EP_TYPE_BULK: Bulk type, EP_TYPE_INTR: Interrupt type</li> <li>• <b>mps</b>: Max Packet Size. This parameter can be a value from 0 to 32K</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 20.2.8 HAL\_HCD\_HC\_Halt

Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDef * hhcd, uint8_t ch_num)</b>
Function Description	Halt a host channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> <li>• <b>ch_num</b>: Channel number. This parameter can be a value from 1 to 15</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 20.2.9 HAL\_HCD\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_HCD_DelInit (HCD_HandleTypeDef * hhcd)</b>
Function Description	DeInitialize the host driver.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 20.2.10 HAL\_HCD\_Msplnit

Function Name	<b>void HAL_HCD_MspInit (HCD_HandleTypeDef * hhcd)</b>
Function Description	Initializes the HCD MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 20.2.11 HAL\_HCD\_MspDeInit

Function Name	<b>void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)</b>
Function Description	DeInitializes HCD MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 20.2.12 HAL\_HCD\_HC\_SubmitRequest

Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)</b>
Function Description	Submit a new URB for processing.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> <li>• <b>ch_num</b>: Channel number. This parameter can be a value from 1 to 15</li> <li>• <b>direction</b>: Channel number. This parameter can be one of these values: 0 : Output / 1 : Input</li> <li>• <b>ep_type</b>: Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type/ EP_TYPE_ISOC: Isochronous type/ EP_TYPE_BULK: Bulk type/ EP_TYPE_INTR: Interrupt type/</li> <li>• <b>token</b>: Endpoint Type. This parameter can be one of these values: 0: HC_PID_SETUP / 1: HC_PID_DATA1</li> <li>• <b>pbuff</b>: pointer to URB data</li> <li>• <b>length</b>: Length of URB data</li> <li>• <b>do_ping</b>: activate do ping protocol (for high speed only). This parameter can be one of these values: 0 : do ping inactive / 1 : do ping active</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 20.2.13 HAL\_HCD\_IRQHandler

Function Name	<b>void HAL_HCD_IRQHandler (HCD_HandleTypeDef * hhcd)</b>
Function Description	This function handles HCD interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 20.2.14 HAL\_HCD\_SOF\_Callback

Function Name	<b>void HAL_HCD_SOF_Callback (HCD_HandleTypeDef * hhcd)</b>
Function Description	SOF callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b>: HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 20.2.15 HAL\_HCD\_Connect\_Callback

Function Name	<b>void HAL_HCD_Connect_Callback (HCD_HandleTypeDef * hhcd)</b>
Function Description	Connexion Event callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b>: HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 20.2.16 HAL\_HCD\_Disconnect\_Callback

Function Name	<b>void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)</b>
Function Description	Disonnexion Event callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b>: HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 20.2.17 HAL\_HCD\_HC\_NotifyURBChange\_Callback

Function Name	<b>void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDef * hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)</b>
Function Description	Notify URB state change callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b>: HCD handle</li><li>• <b>chnum</b>: Channel number. This parameter can be a value from 1 to 15</li><li>• <b>urb_state</b>: This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 20.2.18 HAL\_HCD\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDef *</b>
---------------	---

**hhcd)**

Function Description Start the host driver.

Parameters

- **hhcd**: HCD handle

Return values

- HAL status

**20.2.19 HAL\_HCD\_Stop**

Function Name **HAL\_StatusTypeDef HAL\_HCD\_Stop (HCD\_HandleTypeDef \* hhcd)**

Function Description Stop the host driver.

Parameters

- **hhcd**: HCD handle

Return values

- HAL status

**20.2.20 HAL\_HCD\_ResetPort**

Function Name **HAL\_StatusTypeDef HAL\_HCD\_ResetPort (HCD\_HandleTypeDef \* hhcd)**

Function Description Reset the host port.

Parameters

- **hhcd**: HCD handle

Return values

- HAL status

**20.2.21 HAL\_HCD\_GetState**

Function Name **HCD\_StateTypeDef HAL\_HCD\_GetState (HCD\_HandleTypeDef \* hhcd)**

Function Description Return the HCD state.

Parameters

- **hhcd**: HCD handle

Return values

- HAL state

**20.2.22 HAL\_HCD\_HC\_GetURBState**

Function Name **HCD\_URBStateTypeDef HAL\_HCD\_HC\_GetURBState (HCD\_HandleTypeDef \* hhcd, uint8\_t chnum)**

Function Description Return URB state for a channel.

Parameters

- **hhcd**: HCD handle
- **chnum**: Channel number. This parameter can be a value from 1 to 15

Return values

- URB state. This parameter can be one of these values: URB\_IDLE/ URB\_DONE/ URB\_NOTREADY/ URB\_NYET/ URB\_ERROR/ URB\_STALL/

**20.2.23 HAL\_HCD\_HC\_GetXferCount**

Function Name	<b>uint32_t HAL_HCD_HC_GetXferCount (HCD_HandleTypeDef * hhcd, uint8_t chnum)</b>
Function Description	Return the last host transfer size.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> <li>• <b>chnum</b>: Channel number. This parameter can be a value from 1 to 15</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• last transfer size in byte</li> </ul>

**20.2.24 HAL\_HCD\_HC\_GetState**

Function Name	<b>HCD_HCStateTypeDef HAL_HCD_HC_GetState (HCD_HandleTypeDef * hhcd, uint8_t chnum)</b>
Function Description	Return the Host Channel state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> <li>• <b>chnum</b>: Channel number. This parameter can be a value from 1 to 15</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Host channel state This parameter can be one of the these values: HC_IDLE/ HC_XFRC/ HC_HALTED/ HC_NYET/ HC_NAK/ HC_STALL/ HC_XACTERR/ HC_BBLERR/ HC_DATATGLERR/</li> </ul>

**20.2.25 HAL\_HCD\_GetCurrentFrame**

Function Name	<b>uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)</b>
Function Description	Return the current Host frame number.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Current Host frame number</li> </ul>

**20.2.26 HAL\_HCD\_GetCurrentSpeed**

Function Name	<b>uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)</b>
Function Description	Return the Host enumeration speed.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Enumeration speed</li> </ul>

**20.3 HCD Firmware driver defines**

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



## 20.3.1 HCD

HCD

### ***HCD Exported Macros***

\_\_HAL\_HCD\_ENABLE

\_\_HAL\_HCD\_DISABLE

\_\_HAL\_HCD\_GET\_FLAG

\_\_HAL\_HCD\_CLEAR\_FLAG

\_\_HAL\_HCD\_IS\_INVALID\_INTERRUPT

\_\_HAL\_HCD\_CLEAR\_HC\_INT

\_\_HAL\_HCD\_MASK\_HALT\_HC\_INT

\_\_HAL\_HCD\_UNMASK\_HALT\_HC\_INT

\_\_HAL\_HCD\_MASK\_ACK\_HC\_INT

\_\_HAL\_HCD\_UNMASK\_ACK\_HC\_INT

### ***HCD Instance definition***

IS\_HCD\_ALL\_INSTANCE

### ***HCD Speed***

HCD\_SPEED\_LOW

HCD\_SPEED\_FULL

## 21 HAL I2C Generic Driver

### 21.1 I2C Firmware driver registers structures

#### 21.1.1 I2C\_InitTypeDef

*I2C\_InitTypeDef* is defined in the stm32f1xx\_hal\_i2c.h

##### Data Fields

- *uint32\_t* **ClockSpeed**
- *uint32\_t* **DutyCycle**
- *uint32\_t* **OwnAddress1**
- *uint32\_t* **AddressingMode**
- *uint32\_t* **DualAddressMode**
- *uint32\_t* **OwnAddress2**
- *uint32\_t* **GeneralCallMode**
- *uint32\_t* **NoStretchMode**

##### Field Documentation

- *uint32\_t* **I2C\_InitTypeDef::ClockSpeed** Specifies the clock frequency. This parameter must be set to a value lower than 400kHz
- *uint32\_t* **I2C\_InitTypeDef::DutyCycle** Specifies the I2C fast mode duty cycle. This parameter can be a value of [I2C\\_duty\\_cycle\\_in\\_fast\\_mode](#)
- *uint32\_t* **I2C\_InitTypeDef::OwnAddress1** Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- *uint32\_t* **I2C\_InitTypeDef::AddressingMode** Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of [I2C\\_addressing\\_mode](#)
- *uint32\_t* **I2C\_InitTypeDef::DualAddressMode** Specifies if dual addressing mode is selected. This parameter can be a value of [I2C\\_dual\\_addressing\\_mode](#)
- *uint32\_t* **I2C\_InitTypeDef::OwnAddress2** Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.
- *uint32\_t* **I2C\_InitTypeDef::GeneralCallMode** Specifies if general call mode is selected. This parameter can be a value of [I2C\\_general\\_call\\_addressing\\_mode](#)
- *uint32\_t* **I2C\_InitTypeDef::NoStretchMode** Specifies if nostretch mode is selected. This parameter can be a value of [I2C\\_nostretch\\_mode](#)

#### 21.1.2 I2C\_HandleTypeDef

*I2C\_HandleTypeDef* is defined in the stm32f1xx\_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 uint32_t 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 uint32_t I2C_HandleTypeDef::ErrorCode`

## 21.2 I2C Firmware driver API description

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

### 21.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a `I2C_HandleTypeDef` handle structure, for example: `I2C_HandleTypeDef hi2c;`
2. Initialize the I2C low level resources by implement the `HAL_I2C_MspInit()` API:
  - a. Enable the I2Cx interface clock
  - b. I2C pins configuration
    - Enable the clock for the I2C GPIOs
    - Configure I2C pins as alternate function open-drain
  - c. NVIC configuration if you need to use interrupt process
    - Configure the I2Cx interrupt priority
    - Enable the NVIC I2C IRQ Channel
  - d. DMA Configuration if you need to use DMA process
    - Declare a `DMA_HandleTypeDef` handle structure for the transmit or receive channel
    - Enable the DMAx interface clock using
    - Configure the DMA handle parameters
    - Configure the DMA Tx or Rx channel
    - Associate the initialized DMA handle to the `hi2c` DMA Tx or Rx handle
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel
3. Configure the Communication Speed, Duty cycle, Addressing mode, Own Address1, Dual Addressing mode, Own Address2, General call and Nostretch mode in the `hi2c` Init structure.
4. Initialize the I2C registers by calling the `HAL_I2C_Init()`, configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the 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 operation modes are available within this driver :

### Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using `HAL_I2C_Master_Transmit()`
- Receive in master mode an amount of data in blocking mode using `HAL_I2C_Master_Receive()`
- Transmit in slave mode an amount of data in blocking mode using `HAL_I2C_Slave_Transmit()`
- Receive in slave mode an amount of data in blocking mode using `HAL_I2C_Slave_Receive()`

### Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using `HAL_I2C_Mem_Write()`
- Read an amount of data in blocking mode from a specific memory address using `HAL_I2C_Mem_Read()`

### Interrupt mode IO operation

- The I2C interrupts should have the highest priority in the application in order to make them uninterruptible.
- 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_MasterRxCallback` is executed and user can add his own code by customization of function pointer `HAL_I2C_MasterRxCallback`
- 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_SlaveTxCallback` is executed and user can add his own code by customization of function pointer `HAL_I2C_SlaveTxCallback`
- 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_SlaveRxCallback` is executed and user can add his own code by customization of function pointer `HAL_I2C_SlaveRxCallback`
- 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

- The I2C interrupts should have the highest priority in the application in order to make them uninterruptible.
- 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



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

## I2C Workarounds linked to Silicon Limitation



See ErrataSheet to know full silicon limitation list of your product. (#) Workarounds Implemented inside I2C HAL Driver (##) Wrong data read into data register (Polling and Interrupt mode) (##) Start cannot be generated after a misplaced Stop (##) Some software events must be managed before the current byte is being transferred: Workaround: Use DMA in general, except when the Master is receiving a single byte. For Interrupt mode, I2C should have the highest priority in the application. (##) Mismatch on the "Setup time for a repeated Start condition" timing parameter: Workaround: Reduce the frequency down to 88 kHz or use the I2C Fast-mode if supported by the slave. (##) Data valid time (tVD;DAT) violated without the OVR flag being set: Workaround: If the slave device allows it, use the clock stretching mechanism by programming NoStretchMode = I2C\_NOSTRETCH\_DISABLE in HAL\_I2C\_Init.

### 21.2.2 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:
  - Communication Speed
  - Duty cycle
  - Addressing mode
  - Own Address 1
  - Dual Addressing mode
  - Own Address 2
  - General call mode
  - Nostretch mode
- Call the function HAL\_I2C\_DeInit() to restore the default configuration of the selected I2Cx peripheral.
- [HAL\\_I2C\\_Init\(\)](#)
- [HAL\\_I2C\\_DeInit\(\)](#)
- [HAL\\_I2C\\_MspInit\(\)](#)
- [HAL\\_I2C\\_MspDeInit\(\)](#)

### 21.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

1. There are two modes of transfer:

- Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
    - HAL\_I2C\_Master\_Transmit()
    - HAL\_I2C\_Master\_Receive()
    - HAL\_I2C\_Slave\_Transmit()
    - HAL\_I2C\_Slave\_Receive()
    - HAL\_I2C\_Mem\_Write()
    - HAL\_I2C\_Mem\_Read()
    - HAL\_I2C\_IsDeviceReady()
  3. No-Blocking mode functions with Interrupt are :
    - HAL\_I2C\_Master\_Transmit\_IT()
    - HAL\_I2C\_Master\_Receive\_IT()
    - HAL\_I2C\_Slave\_Transmit\_IT()
    - HAL\_I2C\_Slave\_Receive\_IT()
    - HAL\_I2C\_Mem\_Write\_IT()
    - HAL\_I2C\_Mem\_Read\_IT()
  4. No-Blocking mode functions with DMA are :
    - HAL\_I2C\_Master\_Transmit\_DMA()
    - HAL\_I2C\_Master\_Receive\_DMA()
    - HAL\_I2C\_Slave\_Transmit\_DMA()
    - HAL\_I2C\_Slave\_Receive\_DMA()
    - HAL\_I2C\_Mem\_Write\_DMA()
    - HAL\_I2C\_Mem\_Read\_DMA()
  5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
    - HAL\_I2C\_MemTxCpltCallback()
    - HAL\_I2C\_MemRxCpltCallback()
    - HAL\_I2C\_MasterTxCpltCallback()
    - HAL\_I2C\_MasterRxCpltCallback()
    - HAL\_I2C\_SlaveTxCpltCallback()
    - HAL\_I2C\_SlaveRxCpltCallback()
    - HAL\_I2C\_ErrorCallback()
- [\*\*HAL\\_I2C\\_Master\\_Transmit\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Master\\_Receive\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Slave\\_Transmit\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Slave\\_Receive\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Master\\_Transmit\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Master\\_Receive\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Slave\\_Transmit\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Slave\\_Receive\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Master\\_Transmit\\_DMA\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Master\\_Receive\\_DMA\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Slave\\_Transmit\\_DMA\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Slave\\_Receive\\_DMA\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Mem\\_Write\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Mem\\_Read\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Mem\\_Write\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_I2C\\_Mem\\_Read\\_IT\(\)\*\*](#)

- [HAL\\_I2C\\_Mem\\_Write\\_DMA\(\)](#)
- [HAL\\_I2C\\_Mem\\_Read\\_DMA\(\)](#)
- [HAL\\_I2C\\_IsDeviceReady\(\)](#)

## 21.2.4 Peripheral State and Errors functions

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

- [HAL\\_I2C\\_GetState\(\)](#)
- [HAL\\_I2C\\_GetError\(\)](#)

## 21.2.5 HAL\_I2C\_Init

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

## 21.2.6 HAL\_I2C\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)</b>
Function Description	DeInitializes the I2C peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 21.2.7 HAL\_I2C\_MspInit

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

## 21.2.8 HAL\_I2C\_MspDeInit

Function Name	<b>void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)</b>
Function Description	I2C MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that</li> </ul>



contains the configuration information for the specified I2C.

Return values

- None

### 21.2.9 HAL\_I2C\_Master\_Transmit

**Function Name** `HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)`

**Function Description** Transmits in master mode an amount of data in blocking mode.

**Parameters**

- **hi2c**: : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress**: Target device address
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be sent
- **Timeout**: Timeout duration

**Return values**

- HAL status

### 21.2.10 HAL\_I2C\_Master\_Receive

**Function Name** `HAL_StatusTypeDef HAL_I2C_Master_Receive(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)`

**Function Description** Receives in master mode an amount of data in blocking mode.

**Parameters**

- **hi2c**: : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress**: Target device address
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be sent
- **Timeout**: Timeout duration

**Return values**

- HAL status

### 21.2.11 HAL\_I2C\_Slave\_Transmit

**Function Name** `HAL_StatusTypeDef HAL_I2C_Slave_Transmit(I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)`

**Function Description** Transmits in slave mode an amount of data in blocking mode.

**Parameters**

- **hi2c**: : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be sent
- **Timeout**: Timeout duration

**Return values**

- HAL status

## 21.2.12 HAL\_I2C\_Slave\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive</b> (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"><li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li><li>• <b>pData</b>: Pointer to data buffer</li><li>• <b>Size</b>: Amount of data to be sent</li><li>• <b>Timeout</b>: Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 21.2.13 HAL\_I2C\_Master\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT</b> (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"><li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li><li>• <b>DevAddress</b>: Target device address</li><li>• <b>pData</b>: Pointer to data buffer</li><li>• <b>Size</b>: Amount of data to be sent</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 21.2.14 HAL\_I2C\_Master\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive_IT</b> (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"><li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li><li>• <b>DevAddress</b>: Target device address</li><li>• <b>pData</b>: Pointer to data buffer</li><li>• <b>Size</b>: Amount of data to be sent</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 21.2.15 HAL\_I2C\_Slave\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT</b> (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"> <li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 21.2.16 HAL\_I2C\_Slave\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT</b> (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"> <li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 21.2.17 HAL\_I2C\_Master\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA</b> (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"> <li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li>• <b>DevAddress</b>: Target device address</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 21.2.18 HAL\_I2C\_Master\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA</b> (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"> <li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that</li> </ul>

- 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

### 21.2.19 HAL\_I2C\_Slave\_Transmit\_DMA

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

### 21.2.20 HAL\_I2C\_Slave\_Receive\_DMA

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

### 21.2.21 HAL\_I2C\_Mem\_Write

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

- **Timeout:** Timeout duration
- Return values
  - HAL status

### 21.2.22 HAL\_I2C\_Mem\_Read

- Function Name** `HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)`
- Function Description** Read an amount of data in blocking mode from a specific memory address.
- Parameters**
- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
  - **DevAddress:** Target device address
  - **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

### 21.2.23 HAL\_I2C\_Mem\_Write\_IT

- Function Name** `HAL_StatusTypeDef HAL_I2C_Mem_Write_IT(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)`
- Function Description** Write an amount of data in no-blocking mode with Interrupt to a specific memory address.
- Parameters**
- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
  - **DevAddress:** Target device address
  - **MemAddress:** Internal memory address
  - **MemAddSize:** Size of internal memory address
  - **pData:** Pointer to data buffer
  - **Size:** Amount of data to be sent
- Return values**
- HAL status

### 21.2.24 HAL\_I2C\_Mem\_Read\_IT

- Function Name** `HAL_StatusTypeDef HAL_I2C_Mem_Read_IT(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)`
- Function Description** Read an amount of data in no-blocking mode with Interrupt from a

specific memory address.

Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> <li>• <b>DevAddress</b>: Target device address</li> <li>• <b>MemAddress</b>: Internal memory address</li> <li>• <b>MemAddSize</b>: Size of internal memory address</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 21.2.25 HAL\_I2C\_Mem\_Write\_DMA

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

### 21.2.26 HAL\_I2C\_Mem\_Read\_DMA

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

### 21.2.27 HAL\_I2C\_IsDeviceReady

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

### 21.2.28 HAL\_I2C\_EV\_IRQHandler

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

### 21.2.29 HAL\_I2C\_ER\_IRQHandler

Function Name	<b>void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)</b>
Function Description	This function handles I2C error interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 21.2.30 HAL\_I2C\_MasterTxCpltCallback

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

### 21.2.31 HAL\_I2C\_MasterRxCpltCallback

Function Name	<b>void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)</b>
---------------	---

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

### 21.2.32 HAL\_I2C\_SlaveTxCpltCallback

Function Name	<b>void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)</b>
Function Description	Slave Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 21.2.33 HAL\_I2C\_SlaveRxCpltCallback

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

### 21.2.34 HAL\_I2C\_MemTxCpltCallback

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

### 21.2.35 HAL\_I2C\_MemRxCpltCallback

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



**21.2.36 HAL\_I2C\_ErrorCallback**

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

**21.2.37 HAL\_I2C\_GetState**

Function Name	<b>HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)</b>
Function Description	Returns the I2C state.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

**21.2.38 HAL\_I2C\_GetError**

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

**21.3 I2C Firmware driver defines**

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

**21.3.1 I2C**

I2C

***I2C addressing mode***

I2C\_ADDRESSINGMODE\_7BIT

I2C\_ADDRESSINGMODE\_10BIT

***I2C dual addressing mode***

I2C\_DUALADDRESS\_DISABLE

I2C\_DUALADDRESS\_ENABLE

***I2C Duty Cycle***

I2C\_DUTYCYCLE\_2

I2C\_DUTYCYCLE\_16\_9

**I2C Error Codes**

HAL_I2C_ERROR_NONE	No error
HAL_I2C_ERROR_BERR	BERR error
HAL_I2C_ERROR_ARLO	ARLO error
HAL_I2C_ERROR_AF	AF error
HAL_I2C_ERROR_OVR	OVR error
HAL_I2C_ERROR_DMA	DMA transfer error
HAL_I2C_ERROR_TIMEOUT	Timeout error

**I2C Exported Macros**

`__HAL_I2C_RESET_HANDLE_STATE` **Description:**

- Reset I2C handle state.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.

**Return value:**

- None:

`__HAL_I2C_ENABLE_IT`

**Description:**

- Enable the specified I2C interrupts.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable. This parameter can be one of the following values:
  - `I2C_IT_BUF`: Buffer interrupt enable
  - `I2C_IT_EVT`: Event interrupt enable
  - `I2C_IT_ERR`: Error interrupt enable

**Return value:**

- None:

`__HAL_I2C_DISABLE_IT`

**Description:**

- Disable the specified I2C interrupts.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.
- `__INTERRUPT__`: specifies the interrupt source to disable. This parameter can be one of the following values:
  - `I2C_IT_BUF`: Buffer interrupt enable
  - `I2C_IT_EVT`: Event interrupt enable
  - `I2C_IT_ERR`: Error interrupt enable

**Return value:**

- None:

`__HAL_I2C_GET_IT_SOURCE`

**Description:**

- Checks if the specified I2C interrupt source is enabled or disabled.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.
- `__INTERRUPT__`: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - `I2C_IT_BUF`: Buffer interrupt enable
  - `I2C_IT_EVT`: Event interrupt enable
  - `I2C_IT_ERR`: Error interrupt enable

**Return value:**

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

`__HAL_I2C_GET_FLAG`**Description:**

- Checks whether the specified I2C flag is set or not.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `I2C_FLAG_SMBALERT`: SMBus Alert flag
  - `I2C_FLAG_TIMEOUT`: Timeout or Tlow error flag
  - `I2C_FLAG_PECERR`: PEC error in reception flag
  - `I2C_FLAG_OVR`: Overrun/Underrun flag
  - `I2C_FLAG_AF`: Acknowledge failure flag
  - `I2C_FLAG_ARLO`: Arbitration lost flag
  - `I2C_FLAG_BERR`: Bus error flag
  - `I2C_FLAG_TXE`: Data register empty flag
  - `I2C_FLAG_RXNE`: Data register not empty flag
  - `I2C_FLAG_STOPF`: Stop detection flag
  - `I2C_FLAG_ADD10`: 10-bit header sent flag
  - `I2C_FLAG_BTF`: Byte transfer finished flag
  - `I2C_FLAG_ADDR`: Address sent flag  
Address matched flag
  - `I2C_FLAG_SB`: Start bit flag
  - `I2C_FLAG_DUALF`: Dual flag
  - `I2C_FLAG_SMBHOST`: SMBus host header
  - `I2C_FLAG_SMBDEFAULT`: SMBus default header

- I2C\_FLAG\_GENCALL: General call header flag
- I2C\_FLAG\_TRA: Transmitter/Receiver flag
- I2C\_FLAG\_BUSY: Bus busy flag
- I2C\_FLAG\_MSL: Master/Slave flag

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

**Description:**

- Clears the I2C pending flags which are cleared by writing 0 in a specific bit.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.
- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C\_FLAG\_SMBALERT: SMBus Alert flag
  - I2C\_FLAG\_TIMEOUT: Timeout or Tlow error flag
  - I2C\_FLAG\_PECERR: PEC error in reception flag
  - I2C\_FLAG\_OVR: Overrun/Underrun flag (Slave mode)
  - I2C\_FLAG\_AF: Acknowledge failure flag
  - I2C\_FLAG\_ARLO: Arbitration lost flag (Master mode)
  - I2C\_FLAG\_BERR: Bus error flag

**Return value:**

- None:

**Description:**

- Clears the I2C ADDR pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.

**Return value:**

- None:

**Description:**

- Clears the I2C STOPF pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.

**Return value:**

`__HAL_I2C_CLEAR_FLAG`

`__HAL_I2C_CLEAR_ADDRFLAG`

`__HAL_I2C_CLEAR_STOPFLAG`

\_\_HAL\_I2C\_ENABLE

- None:

**Description:**

- Enable the specified I2C peripheral.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.

**Return value:**

- None:

\_\_HAL\_I2C\_DISABLE

**Description:**

- Disable the specified I2C peripheral.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.

**Return value:**

- None:

***I2C Flag definition***

I2C\_FLAG\_SMBALERT

I2C\_FLAG\_TIMEOUT

I2C\_FLAG\_PECERR

I2C\_FLAG\_OVR

I2C\_FLAG\_AF

I2C\_FLAG\_ARLO

I2C\_FLAG\_BERR

I2C\_FLAG\_TXE

I2C\_FLAG\_RXNE

I2C\_FLAG\_STOPF

I2C\_FLAG\_ADD10

I2C\_FLAG\_BTF

I2C\_FLAG\_ADDR

I2C\_FLAG\_SB

I2C\_FLAG\_DUALF

I2C\_FLAG\_SMBHOST

I2C\_FLAG\_SMBDEFAULT

I2C\_FLAG\_GENCALL

I2C\_FLAG\_TRA

I2C\_FLAG\_BUSY

I2C\_FLAG\_MSL

I2C\_FLAG\_MASK

**I2C general call addressing mode**

I2C\_GENERALCALL\_DISABLE

I2C\_GENERALCALL\_ENABLE

**I2C Interrupt configuration definition**

I2C\_IT\_BUF

I2C\_IT\_EVT

I2C\_IT\_ERR

**I2C Memory Address Size**

I2C\_MEMADD\_SIZE\_8BIT

I2C\_MEMADD\_SIZE\_16BIT

**I2C nostretch mode**

I2C\_NOSTRETCH\_DISABLE

I2C\_NOSTRETCH\_ENABLE

**I2C Private Constants**

I2C\_TIMEOUT\_FLAG

I2C\_TIMEOUT\_ADDR\_SLAVE

I2C\_STANDARD\_MODE\_MAX\_CLK

I2C\_FAST\_MODE\_MAX\_CLK

**I2C Private Macros**

IS\_I2C\_ADDRESSING\_MODE

IS\_I2C\_DUAL\_ADDRESS

IS\_I2C\_GENERAL\_CALL

IS\_I2C\_MEMADD\_SIZE

IS\_I2C\_NO\_STRETCH

IS\_I2C\_OWN\_ADDRESS1

IS\_I2C\_OWN\_ADDRESS2

IS\_I2C\_CLOCK\_SPEED

IS\_I2C\_DUTY\_CYCLE

I2C\_FREQ\_RANGE

I2C\_RISE\_TIME

I2C\_SPEED\_STANDARD

I2C\_SPEED\_FAST

I2C\_SPEED

I2C\_MEM\_ADD\_MSB

I2C\_MEM\_ADD\_LSB

I2C\_7BIT\_ADD\_WRITE

I2C\_7BIT\_ADD\_READ

I2C\_10BIT\_ADDRESS

I2C\_10BIT\_HEADER\_WRITE

I2C\_10BIT\_HEADER\_READ

## 22 HAL I2S Generic Driver

### 22.1 I2S Firmware driver registers structures

#### 22.1.1 I2S\_InitTypeDef

*I2S\_InitTypeDef* is defined in the stm32f1xx\_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](#)

#### 22.1.2 I2S\_HandleTypeDef

*I2S\_HandleTypeDef* is defined in the stm32f1xx\_hal\_i2s.h

##### Data Fields

- *SPI\_TypeDef \* Instance*
- *I2S\_InitTypeDef Init*
- *uint16\_t \* pTxBuffPtr*
- *\_\_IO uint16\_t TxXferSize*
- *\_\_IO uint16\_t TxXferCount*
- *uint16\_t \* pRxBuffPtr*
- *\_\_IO uint16\_t RxXferSize*
- *\_\_IO uint16\_t RxXferCount*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*
- *\_\_IO HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_I2S\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*



### Field Documentation

- *SPI\_TypeDef\* I2S\_HandleTypeDef::Instance*
- *I2S\_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 uint32\_t I2S\_HandleTypeDef::ErrorCode*

## 22.2 I2S Firmware driver API description

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

### 22.2.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a I2S\_HandleTypeDef handle structure.
2. Initialize the I2S low level resources by implement the HAL\_I2S\_MspInit() API:
  - a. Enable the SPIx interface clock.
  - b. I2S pins configuration:
    - Enable the clock for the I2S GPIOs.
    - Configure these I2S pins as alternate function.
  - 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 Channel.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure with the required Tx/Rx parameters.
    - Configure the DMA Tx/Rx Channel.
    - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Channel.
3. Program the 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 \_\_HAL\_I2S\_ENABLE\_IT() and \_\_HAL\_I2S\_DISABLE\_IT() inside the transmit and receive process. The I2SxCLK source is the system clock (provided by the HSI, the HSE or the PLL, and sourcing the AHB clock). For connectivity line devices, the I2SxCLK source can be either SYSCLK or the PLL3 VCO (2 x PLL3CLK) clock in

order to achieve the maximum accuracy. Make sure that either: External clock source is configured after setting correctly the define constant HSE\_VALUE in the stm32f1xx\_hal\_conf.h file.

4. Three mode of operations are available within this driver :

### **Polling mode IO operation**

- Send an amount of data in blocking mode using HAL\_I2S\_Transmit()
- Receive an amount of data in blocking mode using HAL\_I2S\_Receive()

### **Interrupt mode IO operation**

- Send an amount of data in non blocking mode using HAL\_I2S\_Transmit\_IT()
- At transmission end of half transfer HAL\_I2S\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxHalfCpltCallback
- At transmission end of transfer HAL\_I2S\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_I2S\_Receive\_IT()
- At reception end of half transfer HAL\_I2S\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxHalfCpltCallback
- At reception end of transfer HAL\_I2S\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxCpltCallback
- In case of transfer Error, HAL\_I2S\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2S\_ErrorCallback

### **DMA mode IO operation**

- Send an amount of data in non blocking mode (DMA) using HAL\_I2S\_Transmit\_DMA()
- At transmission end of half transfer HAL\_I2S\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxHalfCpltCallback
- At transmission end of transfer HAL\_I2S\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_I2S\_Receive\_DMA()
- At reception end of half transfer HAL\_I2S\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxHalfCpltCallback
- At reception end of transfer HAL\_I2S\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxCpltCallback
- In case of transfer Error, HAL\_I2S\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2S\_ErrorCallback
- Pause the DMA Transfer using HAL\_I2S\_DMAPause()
- Resume the DMA Transfer using HAL\_I2S\_DMAResume()
- Stop the DMA Transfer using HAL\_I2S\_DMAStop()

### I2S HAL driver macros list

Below the list of most used macros in 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

### I2C Workarounds linked to Silicon Limitation



Only the 16-bit mode with no data extension can be used when the I2S is in Master and used the PCM long synchronization mode.

## 22.2.2 Initialization and de-initialization functions

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

- User must Implement `HAL_I2S_MspInit()` function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function `HAL_I2S_Init()` to configure the selected device with the selected configuration:
  - Mode
  - Standard
  - Data Format
  - MCLK Output
  - Audio frequency
  - Polarity
- Call the function `HAL_I2S_DeInit()` to restore the default configuration of the selected I2Sx peripheral.
- [`HAL\_I2S\_Init\(\)`](#)
- [`HAL\_I2S\_DeInit\(\)`](#)
- [`HAL\_I2S\_MspInit\(\)`](#)
- [`HAL\_I2S\_MspDeInit\(\)`](#)

## 22.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2S data transfers.

1. There are two modes of transfer:
  - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.

- No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2S IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
- 2. Blocking mode functions are :
  - HAL\_I2S\_Transmit()
  - HAL\_I2S\_Receive()
- 3. No-Blocking mode functions with Interrupt are :
  - HAL\_I2S\_Transmit\_IT()
  - HAL\_I2S\_Receive\_IT()
- 4. No-Blocking mode functions with DMA are :
  - HAL\_I2S\_Transmit\_DMA()
  - HAL\_I2S\_Receive\_DMA()
- 5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_I2S\_TxCpltCallback()
  - HAL\_I2S\_RxCpltCallback()
  - HAL\_I2S\_ErrorCallback()
- [HAL\\_I2S\\_Transmit\(\)](#)
- [HAL\\_I2S\\_Receive\(\)](#)
- [HAL\\_I2S\\_Transmit\\_IT\(\)](#)
- [HAL\\_I2S\\_Receive\\_IT\(\)](#)
- [HAL\\_I2S\\_Transmit\\_DMA\(\)](#)
- [HAL\\_I2S\\_Receive\\_DMA\(\)](#)
- [HAL\\_I2S\\_DMAPause\(\)](#)
- [HAL\\_I2S\\_DMAResume\(\)](#)
- [HAL\\_I2S\\_DMAStop\(\)](#)
- [HAL\\_I2S\\_IRQHandler\(\)](#)
- [HAL\\_I2S\\_TxHalfCpltCallback\(\)](#)
- [HAL\\_I2S\\_TxCpltCallback\(\)](#)
- [HAL\\_I2S\\_RxHalfCpltCallback\(\)](#)
- [HAL\\_I2S\\_RxCpltCallback\(\)](#)
- [HAL\\_I2S\\_ErrorCallback\(\)](#)

## 22.2.4 Peripheral State and Errors functions

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

- [HAL\\_I2S\\_GetState\(\)](#)
- [HAL\\_I2S\\_GetError\(\)](#)

## 22.2.5 HAL\_I2S\_Init

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

**22.2.6 HAL\_I2S\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)</b>
Function Description	DeInitializes the I2S peripheral.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**22.2.7 HAL\_I2S\_MspInit**

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

**22.2.8 HAL\_I2S\_MspDeInit**

Function Name	<b>void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)</b>
Function Description	I2S MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**22.2.9 HAL\_I2S\_Transmit**

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li><b>pData:</b> a 16-bit pointer to data buffer.</li> <li><b>Size:</b> number of data sample to be sent:</li> <li><b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>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.</li> </ul>

- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

## 22.2.10 HAL\_I2S\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b>: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData</b>: a 16-bit pointer to data buffer.</li> <li>• <b>Size</b>: number of data sample to be sent:</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> <li>• In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continouse way and as the I2S is not disabled at the end of the I2S transaction.</li> </ul>

## 22.2.11 HAL\_I2S\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>
Function Description	Transmit an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b>: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData</b>: a 16-bit pointer to data buffer.</li> <li>• <b>Size</b>: number of data sample to be sent:</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul>

**22.2.12 HAL\_I2S\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive_IT</b> (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Receive an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s</b>: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li><b>pData</b>: a 16-bit pointer to the Receive data buffer.</li> <li><b>Size</b>: number of data sample to be sent:</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>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.</li> <li>The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> <li>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.</li> </ul>

**22.2.13 HAL\_I2S\_Transmit\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit_DMA</b> (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s</b>: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li><b>pData</b>: a 16-bit pointer to the Transmit data buffer.</li> <li><b>Size</b>: number of data sample to be sent:</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>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.</li> <li>The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul>

**22.2.14 HAL\_I2S\_Receive\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive_DMA</b>
---------------	--

---

**(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"> <li><b>hi2s</b>: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li><b>pData</b>: a 16-bit pointer to the Receive data buffer.</li> <li><b>Size</b>: number of data sample to be sent:</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>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.</li> <li>The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul>

### 22.2.15 HAL\_I2S\_DMAPause

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

### 22.2.16 HAL\_I2S\_DMAResume

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

### 22.2.17 HAL\_I2S\_DMAStop

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



## 22.2.18 HAL\_I2S\_IRQHandler

Function Name	<b>void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)</b>
Function Description	This function handles I2S interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2s</b>: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 22.2.19 HAL\_I2S\_TxHalfCpltCallback

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

## 22.2.20 HAL\_I2S\_TxCpltCallback

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

## 22.2.21 HAL\_I2S\_RxHalfCpltCallback

Function Name	<b>void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)</b>
Function Description	Rx Transfer half completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2s</b>: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 22.2.22 HAL\_I2S\_RxCpltCallback

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

**22.2.23 HAL\_I2S\_ErrorCallback**

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

**22.2.24 HAL\_I2S\_GetState**

Function Name	<b>HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * hi2s)</b>
Function Description	Return the I2S state.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

**22.2.25 HAL\_I2S\_GetError**

Function Name	<b>uint32_t HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)</b>
Function Description	Return the I2S error code.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>I2S Error Code</li> </ul>

**22.3 I2S Firmware driver defines**

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

**22.3.1 I2S**

I2S

***I2S Audio Frequency***

I2S\_AUDIOFREQ\_192K

I2S\_AUDIOFREQ\_96K

I2S\_AUDIOFREQ\_48K

I2S\_AUDIOFREQ\_44K

I2S\_AUDIOFREQ\_32K

I2S\_AUDIOFREQ\_22K

I2S\_AUDIOFREQ\_16K

I2S\_AUDIOFREQ\_11K

I2S\_AUDIOFREQ\_8K

I2S\_AUDIOFREQ\_DEFAULT

**I2S Clock Polarity**

I2S\_CPOL\_LOW

I2S\_CPOL\_HIGH

**I2S Data Format**

I2S\_DATAFORMAT\_16B

I2S\_DATAFORMAT\_16B\_EXTENDED

I2S\_DATAFORMAT\_24B

I2S\_DATAFORMAT\_32B

**I2S Error Codes**

HAL\_I2S\_ERROR\_NONE No error

HAL\_I2S\_ERROR\_UDR I2S Underrun error

HAL\_I2S\_ERROR\_OVR I2S Overrun error

HAL\_I2S\_ERROR\_FRE I2S Frame format error

HAL\_I2S\_ERROR\_DMA DMA transfer error

**I2S Exported Macros****\_\_HAL\_I2S\_RESET\_HANDLE\_STATE** **Description:**

- Reset I2S handle state.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.

**Return value:**

- None:

**\_\_HAL\_I2S\_ENABLE****Description:**

- Enable the specified SPI peripheral (in I2S mode).

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.

**Return value:**

- None:

**\_\_HAL\_I2S\_DISABLE****Description:**

- Disable the specified SPI peripheral (in I2S mode).

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.

**Return value:**

- None:

---

**\_\_HAL\_I2S\_ENABLE\_IT****Description:**

- Enable the specified I2S interrupts.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.
- **\_\_INTERRUPT\_\_**: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S\_IT\_TXE: Tx buffer empty interrupt enable
  - I2S\_IT\_RXNE: RX buffer not empty interrupt enable
  - I2S\_IT\_ERR: Error interrupt enable

**Return value:**

- None:

**\_\_HAL\_I2S\_DISABLE\_IT****Description:**

- Disable the specified I2S interrupts.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.
- **\_\_INTERRUPT\_\_**: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S\_IT\_TXE: Tx buffer empty interrupt enable
  - I2S\_IT\_RXNE: RX buffer not empty interrupt enable
  - I2S\_IT\_ERR: Error interrupt enable

**Return value:**

- None:

**\_\_HAL\_I2S\_GET\_IT\_SOURCE****Description:**

- Checks if the specified I2S interrupt source is enabled or disabled.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle. This parameter can be I2S where x: 1, 2, or 3 to select the I2S peripheral.
- **\_\_INTERRUPT\_\_**: specifies the I2S interrupt source to check. This parameter can be one of the following values:
  - I2S\_IT\_TXE: Tx buffer empty interrupt enable
  - I2S\_IT\_RXNE: RX buffer not empty interrupt enable
  - I2S\_IT\_ERR: Error interrupt enable

**Return value:**

- The: new state of **\_\_IT\_\_** (TRUE or FALSE).

**\_\_HAL\_I2S\_GET\_FLAG****Description:**

- Checks whether the specified I2S flag is set or not.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - I2S\_FLAG\_RXNE: Receive buffer not empty flag
  - I2S\_FLAG\_TXE: Transmit buffer empty flag
  - I2S\_FLAG\_UDR: Underrun flag
  - I2S\_FLAG\_OVR: Overrun flag
  - I2S\_FLAG\_CHSIDE: Channel Side flag
  - I2S\_FLAG\_BSY: Busy flag

**Return value:**

- The: new state of **\_\_FLAG\_\_** (TRUE or FALSE).

**\_\_HAL\_I2S\_CLEAR\_OVRFLAG****Description:**

- Clears the I2S OVR pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.

**Return value:**

- None:

**\_\_HAL\_I2S\_CLEAR\_UDRFLAG****Description:**

- Clears the I2S UDR pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.

**Return value:**

- None:

**I2S Flag definition**

I2S\_FLAG\_TXE

I2S\_FLAG\_RXNE

I2S\_FLAG\_UDR

I2S\_FLAG\_OVR

I2S\_FLAG\_FRE

I2S\_FLAG\_CHSIDE

I2S\_FLAG\_BSY

**I2S Interrupt configuration definition**

I2S\_IT\_TXE

I2S\_IT\_RXNE

I2S\_IT\_ERR

***I2S MCLK Output***

I2S\_MCLKOUTPUT\_ENABLE

I2S\_MCLKOUTPUT\_DISABLE

***I2S Mode***

I2S\_MODE\_SLAVE\_TX

I2S\_MODE\_SLAVE\_RX

I2S\_MODE\_MASTER\_TX

I2S\_MODE\_MASTER\_RX

***I2S Private Macros***

IS\_I2S\_MODE

IS\_I2S\_STANDARD

IS\_I2S\_DATA\_FORMAT

IS\_I2S\_MCLK\_OUTPUT

IS\_I2S\_AUDIO\_FREQ

IS\_I2S\_CPOL

***I2S Standard***

I2S\_STANDARD\_PHILIPS

I2S\_STANDARD\_MSB

I2S\_STANDARD\_LSB

I2S\_STANDARD\_PCM\_SHORT

I2S\_STANDARD\_PCM\_LONG

## 23 HAL IRDA Generic Driver

### 23.1 IRDA Firmware driver registers structures

#### 23.1.1 IRDA\_InitTypeDef

*IRDA\_InitTypeDef* is defined in the stm32f1xx\_hal\_irda.h

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint8\_t Prescaler*
- *uint32\_t IrDAMode*

##### Field Documentation

- *uint32\_t IRDA\_InitTypeDef::BaudRate* This member configures the IRDA communication baud rate. The baud rate is computed using the following formula:
  - IntegerDivider = ((PCLKx) / (16 \* (hirda->Init.BaudRate)))
  - FractionalDivider = ((IntegerDivider - ((uint32\_t) IntegerDivider)) \* 16) + 0.5
- *uint32\_t IRDA\_InitTypeDef::WordLength* Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [IRDA\\_Word\\_Length](#)
- *uint32\_t IRDA\_InitTypeDef::Parity* Specifies the parity mode. This parameter can be a value of [IRDA\\_Parity](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- *uint32\_t IRDA\_InitTypeDef::Mode* Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [IRDA\\_Transfer\\_Mode](#)
- *uint8\_t IRDA\_InitTypeDef::Prescaler* Specifies the Prescaler value prescaler value to be programmed in the IrDA low-power Baud Register, for defining pulse width on which burst acceptance/rejection will be decided. This value is used as divisor of system clock to achieve required pulse width.
- *uint32\_t IRDA\_InitTypeDef::IrDAMode* Specifies the IrDA mode This parameter can be a value of [IRDA\\_Low\\_Power](#)

#### 23.1.2 IRDA\_HandleTypeDef

*IRDA\_HandleTypeDef* is defined in the stm32f1xx\_hal\_irda.h

##### Data Fields

- *USART\_TypeDef \* Instance*
- *IRDA\_InitTypeDef Init*
- *uint8\_t \* pTxBuffPtr*
- *uint16\_t TxXferSize*
- *uint16\_t TxXferCount*
- *uint8\_t \* pRxBuffPtr*

- ***uint16\_t RxXferSize***
- ***uint16\_t RxXferCount***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_IRDA\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

#### Field Documentation

- ***USART\_TypeDef\* IRDA\_HandleTypeDef::Instance*** USART registers base address
- ***IRDA\_InitTypeDef IRDA\_HandleTypeDef::Init*** IRDA communication parameters
- ***uint8\_t\* IRDA\_HandleTypeDef::pTxBuffPtr*** Pointer to IRDA Tx transfer Buffer
- ***uint16\_t IRDA\_HandleTypeDef::TxXferSize*** IRDA Tx Transfer size
- ***uint16\_t IRDA\_HandleTypeDef::TxXferCount*** IRDA Tx Transfer Counter
- ***uint8\_t\* IRDA\_HandleTypeDef::pRxBuffPtr*** Pointer to IRDA Rx transfer Buffer
- ***uint16\_t IRDA\_HandleTypeDef::RxXferSize*** IRDA Rx Transfer size
- ***uint16\_t IRDA\_HandleTypeDef::RxXferCount*** IRDA Rx Transfer Counter
- ***DMA\_HandleTypeDef\* IRDA\_HandleTypeDef::hdmatx*** IRDA Tx DMA Handle parameters
- ***DMA\_HandleTypeDef\* IRDA\_HandleTypeDef::hdmarx*** IRDA Rx DMA Handle parameters
- ***HAL\_LockTypeDef IRDA\_HandleTypeDef::Lock*** Locking object
- ***\_\_IO HAL\_IRDA\_StateTypeDef IRDA\_HandleTypeDef::State*** IRDA communication state
- ***\_\_IO uint32\_t IRDA\_HandleTypeDef::ErrorCode*** IRDA Error code

## 23.2 IRDA Firmware driver API description

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

### 23.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a ***IRDA\_HandleTypeDef*** handle structure.
2. Initialize the IRDA low level resources by implementing the ***HAL\_IRDA\_MspInit()*** API:
  - a. Enable the USARTx interface clock.
  - b. IRDA pins configuration:
    - Enable the clock for the IRDA GPIOs.
    - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
  - c. NVIC configuration if you need to use interrupt process (***HAL\_IRDA\_Transmit\_IT()*** and ***HAL\_IRDA\_Receive\_IT()*** APIs):
    - Configure the USARTx interrupt priority.
    - Enable the NVIC USART IRQ handle.
  - d. DMA Configuration if you need to use DMA process (***HAL\_IRDA\_Transmit\_DMA()*** and ***HAL\_IRDA\_Receive\_DMA()*** APIs):
    - Declare a DMA handle structure for the Tx/Rx channel.
    - Enable the DMAx interface clock.



- Configure the declared DMA handle structure with the required Tx/Rx parameters.
  - Configure the DMA Tx/Rx channel.
  - Associate the initialized DMA handle to the IRDA DMA Tx/Rx handle.
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
  - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Parity, IrDA Mode, Prescaler and Mode(Receiver/Transmitter) in the `hirda Init` structure.
  4. Initialize the IRDA registers by calling the `HAL_IRDA_Init()` API:
    - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized `HAL_IRDA_MspInit()` API. The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros `__HAL_IRDA_ENABLE_IT()` and `__HAL_IRDA_DISABLE_IT()` inside the transmit and receive process.
  5. Three operation modes are available within this driver :

### Polling mode IO operation

- Send an amount of data in blocking mode using `HAL_IRDA_Transmit()`
- Receive an amount of data in blocking mode using `HAL_IRDA_Receive()`

### Interrupt mode IO operation

- Send an amount of data in non blocking mode using `HAL_IRDA_Transmit_IT()`
- At transmission end of transfer `HAL_IRDA_TxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_IRDA_TxCpltCallback`
- Receive an amount of data in non blocking mode using `HAL_IRDA_Receive_IT()`
- At reception end of transfer `HAL_IRDA_RxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_IRDA_RxCpltCallback`
- In case of transfer Error, `HAL_IRDA_ErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_IRDA_ErrorCallback`

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using `HAL_IRDA_Transmit_DMA()`
- At transmission end of transfer `HAL_IRDA_TxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_IRDA_TxCpltCallback`
- Receive an amount of data in non blocking mode (DMA) using `HAL_IRDA_Receive_DMA()`
- At reception end of transfer `HAL_IRDA_RxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_IRDA_RxCpltCallback`
- In case of transfer Error, `HAL_IRDA_ErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_IRDA_ErrorCallback`

### IRDA HAL driver macros list

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

- `__HAL_IRDA_ENABLE`: Enable the IRDA peripheral
- `__HAL_IRDA_DISABLE`: Disable the IRDA peripheral
- `__HAL_IRDA_GET_FLAG` : Check whether the specified IRDA flag is set or not
- `__HAL_IRDA_CLEAR_FLAG` : Clear the specified IRDA pending flag
- `__HAL_IRDA_ENABLE_IT`: Enable the specified IRDA interrupt
- `__HAL_IRDA_DISABLE_IT`: Disable the specified IRDA interrupt
- `__HAL_IRDA_GET_IT_SOURCE`: Check whether the specified IRDA interrupt has occurred or not



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

## 23.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in IrDA mode.

- For the asynchronous mode only these parameters can be configured:
  - 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), the possible IRDA frame formats are as listed in [Table 17: "IRDA frame formats"](#)
  - Prescaler: A pulse of width less than two and greater than one PSC period(s) may or may not be rejected. The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).
  - Mode: Receiver/transmitter modes
  - IrDAMode: the IrDA can operate in the Normal mode or in the Low power mode.

**Table 17: IRDA frame formats**

M bit	PCE bit	IRDA frame
0	0	SB   8 bit data   STB
0	1	SB   7 bit data   PB   STB
1	0	SB   9 bit data   STB
1	1	SB   8 bit data   PB   STB

The `HAL_IRDA_Init()` function follows IRDA configuration procedures (details for the procedures are available in reference manuals (RM0008 for STM32F10Xxx MCUs and RM0041 for STM32F100xx MCUs)).

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

## 23.2.3 IO operation functions

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

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
    - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
    - 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 APIs are :
    - HAL\_IRDA\_Transmit()
    - HAL\_IRDA\_Receive()
  3. Non Blocking mode APIs with Interrupt are :
    - HAL\_IRDA\_Transmit\_IT()
    - HAL\_IRDA\_Receive\_IT()
    - HAL\_IRDA\_IRQHandler()
  4. Non Blocking mode functions with DMA are :
    - HAL\_IRDA\_Transmit\_DMA()
    - HAL\_IRDA\_Receive\_DMA()
    - HAL\_IRDA\_DMAPause()
    - HAL\_IRDA\_DMAResume()
    - HAL\_IRDA\_DMAStop()
  5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
    - HAL\_IRDA\_TxHalfCpltCallback()
    - HAL\_IRDA\_TxCpltCallback()
    - HAL\_IRDA\_RxHalfCpltCallback()
    - HAL\_IRDA\_RxCpltCallback()
    - HAL\_IRDA\_ErrorCallback()
- [\*HAL\\_IRDA\\_Transmit\(\)\*](#)
  - [\*HAL\\_IRDA\\_Receive\(\)\*](#)
  - [\*HAL\\_IRDA\\_Transmit\\_IT\(\)\*](#)
  - [\*HAL\\_IRDA\\_Receive\\_IT\(\)\*](#)
  - [\*HAL\\_IRDA\\_Transmit\\_DMA\(\)\*](#)
  - [\*HAL\\_IRDA\\_Receive\\_DMA\(\)\*](#)
  - [\*HAL\\_IRDA\\_DMAPause\(\)\*](#)
  - [\*HAL\\_IRDA\\_DMAResume\(\)\*](#)
  - [\*HAL\\_IRDA\\_DMAStop\(\)\*](#)
  - [\*HAL\\_IRDA\\_IRQHandler\(\)\*](#)
  - [\*HAL\\_IRDA\\_TxCpltCallback\(\)\*](#)
  - [\*HAL\\_IRDA\\_TxHalfCpltCallback\(\)\*](#)
  - [\*HAL\\_IRDA\\_RxCpltCallback\(\)\*](#)
  - [\*HAL\\_IRDA\\_RxHalfCpltCallback\(\)\*](#)
  - [\*HAL\\_IRDA\\_ErrorCallback\(\)\*](#)

## 23.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process

- HAL\_IRDA\_GetState() API can be helpful to check in run-time the state of the IRDA peripheral.
- HAL\_IRDA\_GetError() check in run-time errors that could be occurred during communication.
- [HAL\\_IRDA\\_GetState\(\)](#)
- [HAL\\_IRDA\\_GetError\(\)](#)

## 23.2.5 HAL\_IRDA\_Init

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)</b>
Function Description	Initializes the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 23.2.6 HAL\_IRDA\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)</b>
Function Description	DeInitializes the IRDA peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 23.2.7 HAL\_IRDA\_Msplnit

Function Name	<b>void HAL_IRDA_Msplnit (IRDA_HandleTypeDef * hirda)</b>
Function Description	IRDA MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**23.2.8 HAL\_IRDA\_MspDeInit**

Function Name	<b>void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)</b>
Function Description	IRDA MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li><b>hirda:</b> Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**23.2.9 HAL\_IRDA\_Transmit**

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Sends an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hirda:</b> Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li><b>pData:</b> Pointer to data buffer</li> <li><b>Size:</b> Amount of data to be sent</li> <li><b>Timeout:</b> Specify timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**23.2.10 HAL\_IRDA\_Receive**

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hirda:</b> Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li><b>pData:</b> Pointer to data buffer</li> <li><b>Size:</b> Amount of data to be received</li> <li><b>Timeout:</b> Specify timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**23.2.11 HAL\_IRDA\_Transmit\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)</b>
Function Description	Sends an amount of data in non-blocking mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 23.2.12 HAL\_IRDA\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive_IT</b> (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
Function Description	Receives an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 23.2.13 HAL\_IRDA\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit_DMA</b> (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
Function Description	Sends an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 23.2.14 HAL\_IRDA\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive_DMA</b> (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the IRDA parity is enabled (PCE = 1) the data received</li> </ul>

contain the parity bit.

### 23.2.15 HAL\_IRDA\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)</b>
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 23.2.16 HAL\_IRDA\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)</b>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 23.2.17 HAL\_IRDA\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 23.2.18 HAL\_IRDA\_IRQHandler

Function Name	<b>void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)</b>
Function Description	This function handles IRDA interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 23.2.19 HAL\_IRDA\_TxCpltCallback

Function Name	<b>void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 23.2.20 HAL\_IRDA\_TxHalfCpltCallback

Function Name	<b>void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 23.2.21 HAL\_IRDA\_RxCpltCallback

Function Name	<b>void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 23.2.22 HAL\_IRDA\_RxHalfCpltCallback

Function Name	<b>void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	Rx Half Transfer complete callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 23.2.23 HAL\_IRDA\_ErrorCallback



Function Name	<b>void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	IRDA error callbacks.
Parameters	<ul style="list-style-type: none"> <li><b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 23.2.24 HAL\_IRDA\_GetState

Function Name	<b>HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)</b>
Function Description	Returns the IRDA state.
Parameters	<ul style="list-style-type: none"> <li><b>hirda</b>: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

### 23.2.25 HAL\_IRDA\_GetError

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

## 23.3 IRDA Firmware driver defines

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

### 23.3.1 IRDA

IRDA

#### **IRDA Error Codes**

HAL_IRDA_ERROR_NONE	No error
HAL_IRDA_ERROR_PE	Parity error
HAL_IRDA_ERROR_NE	Noise error
HAL_IRDA_ERROR_FE	frame error
HAL_IRDA_ERROR_ORE	Overrun error
HAL_IRDA_ERROR_DMA	DMA transfer error

#### **IRDA Exported Macros**

**\_\_HAL\_IRDA\_RESET\_HANDLE\_STATE**      **Description:**

- Reset IRDA handle state.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

**Return value:**

- None:

**Description:**

- Flush the IRDA DR register.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `IRDA_FLAG_TXE`: Transmit data register empty flag
  - `IRDA_FLAG_TC`: Transmission Complete flag
  - `IRDA_FLAG_RXNE`: Receive data register not empty flag
  - `IRDA_FLAG_IDLE`: Idle Line detection flag
  - `IRDA_FLAG_ORE`: OverRun Error flag
  - `IRDA_FLAG_NE`: Noise Error flag
  - `IRDA_FLAG_FE`: Framing Error flag
  - `IRDA_FLAG_PE`: Parity Error flag

**Return value:**

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

**Description:**

`__HAL_IRDA_FLUSH_DRREGISTER`

`__HAL_IRDA_GET_FLAG`

`__HAL_IRDA_CLEAR_FLAG`

- Clear the specified IRDA pending flag.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- `__FLAG__`: specifies the flag to check. This parameter can be any combination of the following values:
  - `IRDA_FLAG_TC`: Transmission Complete flag.
  - `IRDA_FLAG_RXNE`: Receive data register not empty flag.

**Return value:**

- None:

**Description:**

- Clear the IRDA PE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None:

**Description:**

- Clear the IRDA FE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None:

**Description:**

- Clear the IRDA NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

`__HAL_IRDA_CLEAR_PEFLAG`

`__HAL_IRDA_CLEAR_FEFLAG`

`__HAL_IRDA_CLEAR_NEFLAG`

**\_\_HAL\_IRDA\_CLEAR\_OREFLAG****Return value:**

- None:

**Description:**

- Clear the IRDA ORE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None:

**\_\_HAL\_IRDA\_CLEAR\_IDLEFLAG****Description:**

- Clear the IRDA IDLE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None:

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

- Enable the specified IRDA interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **\_\_INTERRUPT\_\_**: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data register not empty interrupt
  - IRDA\_IT\_IDLE: Idle line detection interrupt
  - IRDA\_IT\_PE: Parity Error interrupt
  - IRDA\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

**\_\_HAL\_IRDA\_DISABLE\_IT**

- None:

**Description:**

- Disable the specified IRDA interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **\_\_INTERRUPT\_\_**: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data register not empty interrupt
  - IRDA\_IT\_IDLE: Idle line detection interrupt
  - IRDA\_IT\_PE: Parity Error interrupt
  - IRDA\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

**\_\_HAL\_IRDA\_GET\_IT\_SOURCE****Description:**

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

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **\_\_IT\_\_**: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data register not empty interrupt
  - IRDA\_IT\_IDLE: Idle line detection interrupt
  - IRDA\_IT\_ERR: Error interrupt
  - IRDA\_IT\_PE: Parity Error interrupt

**Return value:**

`__HAL_IRDA_ENABLE`

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

**Description:**

- Enable UART/USART associated to IRDA Handle.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

**Return value:**

- None:

`__HAL_IRDA_DISABLE`

**Description:**

- Disable UART/USART associated to IRDA Handle.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

**Return value:**

- None:

**IRDA Flags**

`IRDA_FLAG_TXE`

`IRDA_FLAG_TC`

`IRDA_FLAG_RXNE`

`IRDA_FLAG_IDLE`

`IRDA_FLAG_ORE`

`IRDA_FLAG_NE`

`IRDA_FLAG_FE`

`IRDA_FLAG_PE`

**IRDA Interrupt Definitions**

`IRDA_IT_PE`

`IRDA_IT_TXE`

`IRDA_IT_TC`

`IRDA_IT_RXNE`

`IRDA_IT_IDLE`

`IRDA_IT_LBD`

IRDA\_IT\_CTS

IRDA\_IT\_ERR

**IRDA Low Power**

IRDA\_POWERMODE\_LOWPOWER

IRDA\_POWERMODE\_NORMAL

**IRDA Parity**

IRDA\_PARITY\_NONE

IRDA\_PARITY\_EVEN

IRDA\_PARITY\_ODD

**IRDA Private Constants**

IRDA\_DR\_MASK\_U16\_8DATABITS

IRDA\_DR\_MASK\_U16\_9DATABITS

IRDA\_DR\_MASK\_U8\_7DATABITS

IRDA\_DR\_MASK\_U8\_8DATABITS

**IRDA Private Macros**

IRDA\_CR1\_REG\_INDEX

IRDA\_CR2\_REG\_INDEX

IRDA\_CR3\_REG\_INDEX

IRDA\_DIV

IRDA\_DIVMANT

IRDA\_DIVFRAQ

IRDA\_BRR

IS\_IRDA\_BAUDRATE      The maximum Baud Rate is 115200bps Returns : True or False

IS\_IRDA\_WORD\_LENGTH

IS\_IRDA\_PARITY

IS\_IRDA\_MODE

IS\_IRDA\_POWERMODE

IRDA\_IT\_MASK

**IRDA Transfer Mode**

IRDA\_MODE\_RX

IRDA\_MODE\_TX

IRDA\_MODE\_TX\_RX

**IRDA Word Length**

IRDA\_WORDLENGTH\_8B

IRDA\_WORDLENGTH\_9B

## 24 HAL IWDG Generic Driver

### 24.1 IWDG Firmware driver registers structures

#### 24.1.1 IWDG\_InitTypeDef

*IWDG\_InitTypeDef* is defined in the stm32f1xx\_hal\_iwdg.h

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Reload*

##### Field Documentation

- *uint32\_t IWDG\_InitTypeDef::Prescaler* Select the prescaler of the IWDG. This parameter can be a value of [IWDG\\_Prescaler](#)
- *uint32\_t IWDG\_InitTypeDef::Reload* Specifies the IWDG down-counter reload value. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0xFFFF

#### 24.1.2 IWDG\_HandleTypeDef

*IWDG\_HandleTypeDef* is defined in the stm32f1xx\_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 IWDG Firmware driver API description

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

#### 24.2.1 IWDG specific 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 at 40KHz (LSI): 0.1us / 26.2s . The IWDG timeout may vary due to LSI frequency dispersion. STM32F1xx 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. For more information, please refer to the STM32F1xx Reference manual. Note: LSI Calibration is only available on: High density, XL-density and Connectivity line devices.

### 24.2.2 How to use this driver

- 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.

#### 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_GET_FLAG`: Get the selected IWDG's flag status

### 24.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG\_InitTypeDef and create the associated handle
- Initialize the IWDG MSP
- DeInitialize IWDG MSP
- [\*HAL\\_IWDG\\_Init\(\)\*](#)

- [HAL\\_IWDG\\_Msplnit\(\)](#)

## 24.2.4 IO operation functions

This section provides functions allowing to:

- Start the IWDG.
- Refresh the IWDG.
- [HAL\\_IWDG\\_Start\(\)](#)
- [HAL\\_IWDG\\_Refresh\(\)](#)

## 24.2.5 Peripheral State functions

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

- [HAL\\_IWDG\\_GetState\(\)](#)

## 24.2.6 HAL\_IWDG\_Init

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)</b>
Function Description	Initializes the IWDG according to the specified parameters in the IWDG_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hiwdg</b>: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 24.2.7 HAL\_IWDG\_Msplnit

Function Name	<b>void HAL_IWDG_Msplnit (IWDG_HandleTypeDef * hiwdg)</b>
Function Description	Initializes the IWDG MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hiwdg</b>: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 24.2.8 HAL\_IWDG\_Start

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Start (IWDG_HandleTypeDef * hiwdg)</b>
Function Description	Starts the IWDG.
Parameters	<ul style="list-style-type: none"> <li>• <b>hiwdg</b>: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li> </ul>

Return values

- HAL status

### 24.2.9 HAL\_IWDG\_Refresh

Function Name **HAL\_StatusTypeDef HAL\_IWDG\_Refresh (IWDG\_HandleTypeDef \* hiwdg)**

Function Description Refreshes the IWDG.

Parameters

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

Return values

- HAL status

### 24.2.10 HAL\_IWDG\_GetState

Function Name **HAL\_IWDG\_StateTypeDef HAL\_IWDG\_GetState (IWDG\_HandleTypeDef \* hiwdg)**

Function Description Returns the IWDG state.

Parameters

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

Return values

- HAL state

## 24.3 IWDG Firmware driver defines

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

### 24.3.1 IWDG

IWDG

#### *IWDG Exported Macros*

**\_\_HAL\_IWDG\_RESET\_HANDLE\_STATE** **Description:**

- Reset IWDG handle state.

**Parameters:**

- **\_\_HANDLE\_\_**: IWDG handle.

**Return value:**

- None:

**\_\_HAL\_IWDG\_START**

**Description:**

- Enables the IWDG peripheral.

**Parameters:**

- **\_\_HANDLE\_\_**: IWDG handle

**Return value:**

**\_\_HAL\_IWDG\_RELOAD\_COUNTER**

- None:

**Description:**

- Reloads IWDG counter with value defined in the reload register (write access to IWDG\_PR and IWDG\_RLR registers disabled).

**Parameters:**

- \_\_HANDLE\_\_: IWDG handle

**Return value:**

- None:

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

- Gets the selected IWDG's flag status.

**Parameters:**

- \_\_HANDLE\_\_: IWDG handle
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - IWDG\_FLAG\_PVU: Watchdog counter reload value update flag
  - IWDG\_FLAG\_RVU: Watchdog counter prescaler value flag

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

**IWDG Flag definition**

IWDG\_FLAG\_PVU      Watchdog counter prescaler value update Flag

IWDG\_FLAG\_RVU      Watchdog counter reload value update Flag

**IWDG Prescaler**

IWDG\_PRESCALER\_4      IWDG prescaler set to 4

IWDG\_PRESCALER\_8      IWDG prescaler set to 8

IWDG\_PRESCALER\_16      IWDG prescaler set to 16

IWDG\_PRESCALER\_32      IWDG prescaler set to 32

IWDG\_PRESCALER\_64      IWDG prescaler set to 64

IWDG\_PRESCALER\_128      IWDG prescaler set to 128

IWDG\_PRESCALER\_256      IWDG prescaler set to 256

**IWDG Private Constants**

IWDG\_DEFAULT\_TIMEOUT

**IWDG Private Macros**

IWDG\_ENABLE\_WRITE\_ACCESS

**Description:**

- Enables write access to IWDG\_PR and

IWDG\_RLR registers.

**Parameters:**

- `__HANDLE__`: IWDG handle

**Return value:**

- None:

`IWDG_DISABLE_WRITE_ACCESS`

**Description:**

- Disables write access to IWDG\_PR and IWDG\_RLR registers.

**Parameters:**

- `__HANDLE__`: IWDG handle

**Return value:**

- None:

`IS_IWDG_PRESCALER`

`IS_IWDG_RELOAD`

***IWDG Registers BitMask***

`IWDG_KR_KEY_RELOAD` IWDG Reload Counter Enable

`IWDG_KR_KEY_ENABLE` IWDG Peripheral Enable

`IWDG_KR_KEY_EWA` IWDG KR Write Access Enable

`IWDG_KR_KEY_DWA` IWDG KR Write Access Disable

## 25 HAL NAND Generic Driver

### 25.1 NAND Firmware driver registers structures

#### 25.1.1 NAND\_IDTypeDef

**NAND\_IDTypeDef** is defined in the stm32f1xx\_hal\_nand.h

##### Data Fields

- *uint8\_t Maker\_Id*
- *uint8\_t Device\_Id*
- *uint8\_t Third\_Id*
- *uint8\_t Fourth\_Id*

##### Field Documentation

- *uint8\_t NAND\_IDTypeDef::Maker\_Id*
- *uint8\_t NAND\_IDTypeDef::Device\_Id*
- *uint8\_t NAND\_IDTypeDef::Third\_Id*
- *uint8\_t NAND\_IDTypeDef::Fourth\_Id*

#### 25.1.2 NAND\_AddressTypeDef

**NAND\_AddressTypeDef** is defined in the stm32f1xx\_hal\_nand.h

##### Data Fields

- *uint16\_t Page*
- *uint16\_t Zone*
- *uint16\_t Block*

##### Field Documentation

- *uint16\_t NAND\_AddressTypeDef::Page* NAND memory Page address
- *uint16\_t NAND\_AddressTypeDef::Zone* NAND memory Zone address
- *uint16\_t NAND\_AddressTypeDef::Block* NAND memory Block address

#### 25.1.3 NAND\_InfoTypeDef

**NAND\_InfoTypeDef** is defined in the stm32f1xx\_hal\_nand.h

##### Data Fields

- *uint32\_t PageSize*
- *uint32\_t SpareAreaSize*
- *uint32\_t BlockSize*
- *uint32\_t BlockNbr*
- *uint32\_t ZoneSize*

**Field Documentation**

- ***uint32\_t NAND\_InfoTypeDef::PageSize*** NAND memory page (without spare area) size measured in K. bytes
- ***uint32\_t NAND\_InfoTypeDef::SpareAreaSize*** NAND memory spare area size measured in K. bytes
- ***uint32\_t NAND\_InfoTypeDef::BlockSize*** NAND memory block size number of pages
- ***uint32\_t NAND\_InfoTypeDef::BlockNbr*** NAND memory number of blocks
- ***uint32\_t NAND\_InfoTypeDef::ZoneSize*** NAND memory zone size measured in number of blocks

**25.1.4 NAND\_HandleTypeDef**

***NAND\_HandleTypeDef*** is defined in the stm32f1xx\_hal\_nand.h

**Data Fields**

- ***FSMC\_NAND\_TypeDef \* Instance***
- ***FSMC\_NAND\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_NAND\_StateTypeDef State***
- ***NAND\_InfoTypeDef Info***

**Field Documentation**

- ***FSMC\_NAND\_TypeDef\* NAND\_HandleTypeDef::Instance*** Register base address
- ***FSMC\_NAND\_InitTypeDef NAND\_HandleTypeDef::Init*** NAND device control configuration parameters
- ***HAL\_LockTypeDef NAND\_HandleTypeDef::Lock*** NAND locking object
- ***\_\_IO HAL\_NAND\_StateTypeDef NAND\_HandleTypeDef::State*** NAND device access state
- ***NAND\_InfoTypeDef NAND\_HandleTypeDef::Info*** NAND characteristic information structure

**25.2 NAND Firmware driver API description**

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

**25.2.1 How to use this driver**

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FSMC/FSMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function ***HAL\_NAND\_Init()*** with control and timing parameters for both common and attribute spaces.
- Read NAND flash memory maker and device IDs using the function ***HAL\_NAND\_Read\_ID()***. The read information is stored in the ***NAND\_ID\_TypeDef*** structure declared by the function caller.
- Access NAND flash memory by read/write operations using the functions ***HAL\_NAND\_Read\_Page()***/***HAL\_NAND\_Read\_SpareArea()***, ***HAL\_NAND\_Write\_Page()***/***HAL\_NAND\_Write\_SpareArea()*** to read/write

page(s)/spare area(s). These functions use specific device information (Block, page size..) predefined by the user in the HAL\_NAND\_Info\_TypeDef structure. The read/write address information is contained by the Nand\_Address\_Typedef structure passed as parameter.

- Perform NAND flash Reset chip operation using the function HAL\_NAND\_Reset().
- Perform NAND flash erase block operation using the function HAL\_NAND\_Erase\_Block(). The erase block address information is contained in the Nand\_Address\_Typedef structure passed as parameter.
- Read the NAND flash status operation using the function HAL\_NAND\_Read\_Status().
- You can also control the NAND device by calling the control APIs HAL\_NAND\_ECC\_Enable()/ HAL\_NAND\_ECC\_Disable() to respectively enable/disable the ECC code correction feature or the function HAL\_NAND\_GetECC() to get the ECC correction code.
- You can monitor the NAND device HAL state by calling the function HAL\_NAND\_GetState()



This driver is a set of generic APIs which handle standard NAND flash operations. If a NAND flash device contains different operations and/or implementations, it should be implemented separately.

### 25.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

- [\*HAL\\_NAND\\_Init\(\)\*](#)
- [\*HAL\\_NAND\\_DeInit\(\)\*](#)
- [\*HAL\\_NAND\\_MspInit\(\)\*](#)
- [\*HAL\\_NAND\\_MspDeInit\(\)\*](#)
- [\*HAL\\_NAND\\_IRQHandler\(\)\*](#)
- [\*HAL\\_NAND\\_ITCallback\(\)\*](#)

### 25.2.3 NAND Input and Output functions

This section provides functions allowing to use and control the NAND memory

- [\*HAL\\_NAND\\_Read\\_ID\(\)\*](#)
- [\*HAL\\_NAND\\_Reset\(\)\*](#)
- [\*HAL\\_NAND\\_Read\\_Page\(\)\*](#)
- [\*HAL\\_NAND\\_Write\\_Page\(\)\*](#)
- [\*HAL\\_NAND\\_Read\\_SpareArea\(\)\*](#)
- [\*HAL\\_NAND\\_Write\\_SpareArea\(\)\*](#)
- [\*HAL\\_NAND\\_Erase\\_Block\(\)\*](#)
- [\*HAL\\_NAND\\_Read\\_Status\(\)\*](#)
- [\*HAL\\_NAND\\_Address\\_Inc\(\)\*](#)

### 25.2.4 NAND Control functions

This subsection provides a set of functions allowing to control dynamically the NAND interface.

- [\*HAL\\_NAND\\_ECC\\_Enable\(\)\*](#)



- [HAL\\_NAND\\_ECC\\_Disable\(\)](#)
- [HAL\\_NAND\\_GetECC\(\)](#)

### 25.2.5 NAND State functions

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

- [HAL\\_NAND\\_GetState\(\)](#)
- [HAL\\_NAND\\_Read\\_Status\(\)](#)

### 25.2.6 HAL\_NAND\_Init

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FSMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FSMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)</b>
Function Description	Perform NAND memory Initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li>• <b>ComSpace_Timing</b>: pointer to Common space timing structure</li> <li>• <b>AttSpace_Timing</b>: pointer to Attribute space timing structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 25.2.7 HAL\_NAND\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)</b>
Function Description	Perform NAND memory De-Initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 25.2.8 HAL\_NAND\_Msplnit

Function Name	<b>void HAL_NAND_Msplnit (NAND_HandleTypeDef * hnand)</b>
Function Description	NAND MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 25.2.9 HAL\_NAND\_MspDeInit

Function Name	<b>void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnand)</b>
Function Description	NAND MSP DeInit.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 25.2.10 HAL\_NAND\_IRQHandler

Function Name	<b>void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnand)</b>
Function Description	This function handles NAND device interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 25.2.11 HAL\_NAND\_ITCallback

Function Name	<b>void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnand)</b>
Function Description	NAND interrupt feature callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 25.2.12 HAL\_NAND\_Read\_ID

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)</b>
Function Description	Read the NAND memory electronic signature.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li><li>• <b>pNAND_ID</b>: NAND ID structure</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 25.2.13 HAL\_NAND\_Reset

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnand)</b>
Function Description	NAND memory reset.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li></ul>

Return values

- HAL status

## 25.2.14 HAL\_NAND\_Read\_Page

**Function Name** `HAL_StatusTypeDef HAL_NAND_Read_Page(NAND_HandleTypeDef * hnd, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)`

**Function Description** Read Page(s) from NAND memory block.

**Parameters**

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

**Return values**

- HAL status

## 25.2.15 HAL\_NAND\_Write\_Page

**Function Name** `HAL_StatusTypeDef HAL_NAND_Write_Page(NAND_HandleTypeDef * hnd, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)`

**Function Description** Write Page(s) to NAND memory block.

**Parameters**

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

**Return values**

- HAL status

## 25.2.16 HAL\_NAND\_Read\_SpareArea

**Function Name** `HAL_StatusTypeDef HAL_NAND_Read_SpareArea(NAND_HandleTypeDef * hnd, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)`

**Function Description** Read Spare area(s) from NAND memory.

**Parameters**

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

**Return values**

- HAL status

## 25.2.17 HAL\_NAND\_Write\_SpareArea

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Write_SpareArea</b> (NAND_HandleTypeDef * h NAND, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)
Function Description	Write Spare area(s) to NAND memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>h NAND</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li>• <b>pAddress</b>: : pointer to NAND address structure</li> <li>• <b>pBuffer</b>: : pointer to source buffer to write</li> <li>• <b>NumSpareAreaTowrite</b>: : number of spare areas to write to block</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 25.2.18 HAL\_NAND\_Erase\_Block

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Erase_Block</b> (NAND_HandleTypeDef * h NAND, NAND_AddressTypeDef * pAddress)
Function Description	NAND memory Block erase.
Parameters	<ul style="list-style-type: none"> <li>• <b>h NAND</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li>• <b>pAddress</b>: : pointer to NAND address structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 25.2.19 HAL\_NAND\_Read\_Status

Function Name	<b>uint32_t HAL_NAND_Read_Status</b> (NAND_HandleTypeDef * h NAND)
Function Description	NAND memory read status.
Parameters	<ul style="list-style-type: none"> <li>• <b>h NAND</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• NAND status</li> </ul>

### 25.2.20 HAL\_NAND\_Address\_Inc

Function Name	<b>uint32_t HAL_NAND_Address_Inc</b> (NAND_HandleTypeDef * h NAND, NAND_AddressTypeDef * pAddress)
Function Description	Increment the NAND memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>h NAND</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li>• <b>pAddress</b>: pointer to NAND address structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• The new status of the increment address operation. It can be: NAND_VALID_ADDRESS: When the new address is valid addressNAND_INVALID_ADDRESS: When the new address</li> </ul>

is invalid address

### 25.2.21 HAL\_NAND\_ECC\_Enable

Function Name	<b>HAL_StatusTypeDef HAL_NAND_ECC_Enable (NAND_HandleTypeDef * hnand)</b>
Function Description	Enables dynamically NAND ECC feature.
Parameters	<ul style="list-style-type: none"> <li><b>hnand:</b> pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 25.2.22 HAL\_NAND\_ECC\_Disable

Function Name	<b>HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)</b>
Function Description	Disables dynamically FSMC_NAND ECC feature.
Parameters	<ul style="list-style-type: none"> <li><b>hnand:</b> pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 25.2.23 HAL\_NAND\_GetECC

Function Name	<b>HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hnand, uint32_t * ECCval, uint32_t Timeout)</b>
Function Description	Disables dynamically NAND ECC feature.
Parameters	<ul style="list-style-type: none"> <li><b>hnand:</b> pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li><b>ECCval:</b> pointer to ECC value</li> <li><b>Timeout:</b> maximum timeout to wait</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 25.2.24 HAL\_NAND\_GetState

Function Name	<b>HAL_NAND_StateTypeDef HAL_NAND_GetState (NAND_HandleTypeDef * hnand)</b>
Function Description	return the NAND state
Parameters	<ul style="list-style-type: none"> <li><b>hnand:</b> pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

## 25.2.25 HAL\_NAND\_Read\_Status

Function Name	uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hnand)
Function Description	NAND memory read status.
Parameters	<ul style="list-style-type: none"> <li><b>hnand</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>NAND status</li> </ul>

## 25.3 NAND Firmware driver defines

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

### 25.3.1 NAND

NAND

#### **NAND Exported Macros**

<code>__HAL_NAND_RESET_HANDLE_STATE</code>	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset NAND handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><code>__HANDLE__</code>: specifies the NAND handle.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None:</li> </ul>
--	--

#### **NAND Private Constants**

NAND\_DEVICE1  
 NAND\_DEVICE2  
 NAND\_WRITE\_TIMEOUT  
 CMD\_AREA  
 ADDR\_AREA  
 NAND\_CMD\_AREA\_A  
 NAND\_CMD\_AREA\_B  
 NAND\_CMD\_AREA\_C  
 NAND\_CMD\_AREA\_TRUE1  
 NAND\_CMD\_WRITE0  
 NAND\_CMD\_WRITE\_TRUE1  
 NAND\_CMD\_ERASE0  
 NAND\_CMD\_ERASE1  
 NAND\_CMD\_READID  
 NAND\_CMD\_STATUS

NAND\_CMD\_LOCK\_STATUS

NAND\_CMD\_RESET

NAND\_VALID\_ADDRESS

NAND\_INVALID\_ADDRESS

NAND\_TIMEOUT\_ERROR

NAND\_BUSY

NAND\_ERROR

NAND\_READY

**NAND Private Macros**

\_\_ARRAY\_ADDRESS

**Description:**

- NAND memory address computation.

**Parameters:**

- \_\_ADDRESS\_\_: NAND memory address.
- \_\_HANDLE\_\_: : NAND handle.

**Return value:**

- NAND: Raw address value

\_\_ADDR\_1st\_CYCLE

**Description:**

- NAND memory address cycling.

**Parameters:**

- \_\_ADDRESS\_\_: NAND memory address.

**Return value:**

- NAND: address cycling value.

\_\_ADDR\_2nd\_CYCLE

\_\_ADDR\_3rd\_CYCLE

\_\_ADDR\_4th\_CYCLE

## 26 HAL NOR Generic Driver

### 26.1 NOR Firmware driver registers structures

#### 26.1.1 NOR\_IDTypeDef

*NOR\_IDTypeDef* is defined in the stm32f1xx\_hal\_nor.h

##### Data Fields

- *uint16\_t Manufacturer\_Code*
- *uint16\_t Device\_Code1*
- *uint16\_t Device\_Code2*
- *uint16\_t Device\_Code3*

##### Field Documentation

- *uint16\_t NOR\_IDTypeDef::Manufacturer\_Code* Defines the device's manufacturer code used to identify the memory
- *uint16\_t NOR\_IDTypeDef::Device\_Code1*
- *uint16\_t NOR\_IDTypeDef::Device\_Code2*
- *uint16\_t NOR\_IDTypeDef::Device\_Code3* Defines the device's codes used to identify the memory. These codes can be accessed by performing read operations with specific control signals and addresses set. They can also be accessed by issuing an Auto Select command

#### 26.1.2 NOR\_CFITypeDef

*NOR\_CFITypeDef* is defined in the stm32f1xx\_hal\_nor.h

##### Data Fields

- *uint16\_t CFI\_1*
- *uint16\_t CFI\_2*
- *uint16\_t CFI\_3*
- *uint16\_t CFI\_4*

##### Field Documentation

- *uint16\_t NOR\_CFITypeDef::CFI\_1* < Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory
- *uint16\_t NOR\_CFITypeDef::CFI\_2*
- *uint16\_t NOR\_CFITypeDef::CFI\_3*
- *uint16\_t NOR\_CFITypeDef::CFI\_4*

#### 26.1.3 NOR\_HandleTypeDef

*NOR\_HandleTypeDef* is defined in the stm32f1xx\_hal\_nor.h

##### Data Fields



- ***FSMC\_NORSRAM\_TypeDef \* Instance***
- ***FSMC\_NORSRAM\_EXTENDED\_TypeDef \* Extended***
- ***FSMC\_NORSRAM\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_NOR\_StateTypeDef State***

#### Field Documentation

- ***FSMC\_NORSRAM\_TypeDef\* NOR\_HandleTypeDef::Instance*** Register base address
- ***FSMC\_NORSRAM\_EXTENDED\_TypeDef\* NOR\_HandleTypeDef::Extended*** Extended mode register base address
- ***FSMC\_NORSRAM\_InitTypeDef NOR\_HandleTypeDef::Init*** NOR device control configuration parameters
- ***HAL\_LockTypeDef NOR\_HandleTypeDef::Lock*** NOR locking object
- ***\_\_IO HAL\_NOR\_StateTypeDef NOR\_HandleTypeDef::State*** NOR device access state

## 26.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

### 26.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FSMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function `HAL_NOR_Init()` with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function `HAL_NOR_Read_ID()`. The read information is stored in the `NOR_ID_TypeDef` structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions `HAL_NOR_Read()`, `HAL_NOR_Program()`.
- Perform NOR flash erase block/chip operations using the functions `HAL_NOR_Erase_Block()` and `HAL_NOR_Erase_Chip()`.
- Read the NOR flash CFI (common flash interface) IDs using the function `HAL_NOR_Read_CFI()`. The read information is stored in the `NOR_CFI_TypeDef` structure declared by the function caller.
- You can also control the NOR device by calling the control APIs `HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable()` to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function `HAL_NOR_GetState()`



This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

### NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- `__NOR_WRITE` : NOR memory write data to specified address

## 26.2.2 NOR Initialization and de\_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

- [`HAL\_NOR\_Init\(\)`](#)
- [`HAL\_NOR\_DeInit\(\)`](#)
- [`HAL\_NOR\_MspInit\(\)`](#)
- [`HAL\_NOR\_MspDeInit\(\)`](#)
- [`HAL\_NOR\_MspWait\(\)`](#)

## 26.2.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

- [`HAL\_NOR\_Read\_ID\(\)`](#)
- [`HAL\_NOR\_ReturnToReadMode\(\)`](#)
- [`HAL\_NOR\_Read\(\)`](#)
- [`HAL\_NOR\_Program\(\)`](#)
- [`HAL\_NOR\_ReadBuffer\(\)`](#)
- [`HAL\_NOR\_ProgramBuffer\(\)`](#)
- [`HAL\_NOR\_Erase\_Block\(\)`](#)
- [`HAL\_NOR\_Erase\_Chip\(\)`](#)
- [`HAL\_NOR\_Read\_CFI\(\)`](#)

## 26.2.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

- [`HAL\_NOR\_WriteOperation\_Enable\(\)`](#)
- [`HAL\_NOR\_WriteOperation\_Disable\(\)`](#)

## 26.2.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

- [`HAL\_NOR\_GetState\(\)`](#)
- [`HAL\_NOR\_GetStatus\(\)`](#)

## 26.2.6 HAL\_NOR\_Init

Function Name	<code>HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FSMC_NORSRAM_TimingTypeDef * Timing, FSMC_NORSRAM_TimingTypeDef * ExtTiming)</code>
---------------	---

Function Description	Perform the NOR memory Initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b>: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>Timing</b>: pointer to NOR control timing structure</li> <li>• <b>ExtTiming</b>: pointer to NOR extended mode timing structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 26.2.7 HAL\_NOR\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)</b>
Function Description	Perform NOR memory De-Initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b>: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 26.2.8 HAL\_NOR\_MspInit

Function Name	<b>void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)</b>
Function Description	NOR MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b>: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 26.2.9 HAL\_NOR\_MspDeInit

Function Name	<b>void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)</b>
Function Description	NOR MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b>: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 26.2.10 HAL\_NOR\_MspWait

Function Name	<b>void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)</b>
Function Description	NOR BSP Wait fro Ready/Busy signal.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b>: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>Timeout</b>: Maximum timeout value</li> </ul>

Return values

- None

### 26.2.11 HAL\_NOR\_Read\_ID

**Function Name** HAL\_StatusTypeDef HAL\_NOR\_Read\_ID  
(NOR\_HandleTypeDef \* hnor, NOR\_IDTypeDef \* pNOR\_ID)

**Function Description** Read NOR flash IDs.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR\_ID**: : pointer to NOR ID structure

**Return values**

- HAL status

### 26.2.12 HAL\_NOR\_ReturnToReadMode

**Function Name** HAL\_StatusTypeDef HAL\_NOR\_ReturnToReadMode  
(NOR\_HandleTypeDef \* hnor)

**Function Description** Returns the NOR memory to Read mode.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.

**Return values**

- HAL status

### 26.2.13 HAL\_NOR\_Read

**Function Name** HAL\_StatusTypeDef HAL\_NOR\_Read (NOR\_HandleTypeDef \*  
hnor, uint32\_t \* pAddress, uint16\_t \* pData)

**Function Description** Read data from NOR memory.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress**: pointer to Device address
- **pData**: : pointer to read data

**Return values**

- HAL status

### 26.2.14 HAL\_NOR\_Program

**Function Name** HAL\_StatusTypeDef HAL\_NOR\_Program  
(NOR\_HandleTypeDef \* hnor, uint32\_t \* pAddress, uint16\_t \*  
pData)

**Function Description** Program data to NOR memory.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress**: Device address
- **pData**: : pointer to the data to write

Return values

- HAL status

### 26.2.15 HAL\_NOR\_ReadBuffer

**Function Name** **HAL\_StatusTypeDef HAL\_NOR\_ReadBuffer**  
(NOR\_HandleTypeDef \* hnor, uint32\_t uwAddress, uint16\_t \* pData, uint32\_t uwBufferSize)

**Function Description** Reads a block of data from the FSMC NOR memory.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
- **uwAddress**: NOR memory internal address to read from.
- **pData**: pointer to the buffer that receives the data read from the NOR memory.
- **uwBufferSize**: : number of Half word to read.

Return values

- HAL status

### 26.2.16 HAL\_NOR\_ProgramBuffer

**Function Name** **HAL\_StatusTypeDef HAL\_NOR\_ProgramBuffer**  
(NOR\_HandleTypeDef \* hnor, uint32\_t uwAddress, uint16\_t \* pData, uint32\_t uwBufferSize)

**Function Description** Writes a half-word buffer to the FSMC NOR memory.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
- **uwAddress**: NOR memory internal address from which the data
- **pData**: pointer to source data buffer.
- **uwBufferSize**: number of Half words to write.

Return values

- HAL status

**Notes**

- Some NOR memory need Address aligned to xx bytes (can be aligned to 64 bytes boundary for example).
- The maximum buffer size allowed is NOR memory dependent (can be 64 Bytes max for example).

### 26.2.17 HAL\_NOR\_Erase\_Block

**Function Name** **HAL\_StatusTypeDef HAL\_NOR\_Erase\_Block**  
(NOR\_HandleTypeDef \* hnor, uint32\_t BlockAddress, uint32\_t Address)

**Function Description** Erase the specified block of the NOR memory.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
- **BlockAddress**: : Block to erase address
- **Address**: Device address

Return values

- HAL status

### 26.2.18 HAL\_NOR\_Erase\_Chip

**Function Name** HAL\_StatusTypeDef HAL\_NOR\_Erase\_Chip  
(NOR\_HandleTypeDef \* hnor, uint32\_t Address)

**Function Description** Erase the entire NOR chip.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address**: : Device address

**Return values**

- HAL status

### 26.2.19 HAL\_NOR\_Read\_CFI

**Function Name** HAL\_StatusTypeDef HAL\_NOR\_Read\_CFI  
(NOR\_HandleTypeDef \* hnor, NOR\_CFITypeDef \* pNOR\_CFI)

**Function Description** Read NOR flash CFI IDs.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR\_CFI**: : pointer to NOR CFI IDs structure

**Return values**

- HAL status

### 26.2.20 HAL\_NOR\_WriteOperation\_Enable

**Function Name** HAL\_StatusTypeDef HAL\_NOR\_WriteOperation\_Enable  
(NOR\_HandleTypeDef \* hnor)

**Function Description** Enables dynamically NOR write operation.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.

**Return values**

- HAL status

### 26.2.21 HAL\_NOR\_WriteOperation\_Disable

**Function Name** HAL\_StatusTypeDef HAL\_NOR\_WriteOperation\_Disable  
(NOR\_HandleTypeDef \* hnor)

**Function Description** Disables dynamically NOR write operation.

**Parameters**

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.

**Return values**

- HAL status

## 26.2.22 HAL\_NOR\_GetState

Function Name	<b>HAL_NOR_StateTypeDef HAL_NOR_GetState (NOR_HandleTypeDef * hnor)</b>
Function Description	return the NOR controller state
Parameters	<ul style="list-style-type: none"> <li><b>hnor</b>: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>NOR controller state</li> </ul>

## 26.2.23 HAL\_NOR\_GetStatus

Function Name	<b>NOR_StatusTypeDef HAL_NOR_GetStatus (NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)</b>
Function Description	Returns the NOR operation status.
Parameters	<ul style="list-style-type: none"> <li><b>hnor</b>: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li><b>Address</b>: Device address</li> <li><b>Timeout</b>: NOR programming Timeout</li> </ul>
Return values	<ul style="list-style-type: none"> <li>NOR_Status The returned value can be: NOR_SUCCESS, NOR_ERROR or NOR_TIMEOUT</li> </ul>

## 26.3 NOR Firmware driver defines

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

### 26.3.1 NOR

NOR

#### ***NOR Exported Macros***

<b>__HAL_NOR_RESET_HANDLE_STATE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset NOR handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__</b>: NOR handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None:</li> </ul>
-------------------------------------	---

#### ***NOR Private Constants***

NOR\_CMD\_ADDRESS\_FIRST  
 NOR\_CMD\_ADDRESS\_FIRST\_CFI  
 NOR\_CMD\_ADDRESS\_SECOND  
 NOR\_CMD\_ADDRESS\_THIRD  
 NOR\_CMD\_ADDRESS\_FOURTH

NOR\_CMD\_ADDRESS\_FIFTH  
NOR\_CMD\_ADDRESS\_SIXTH  
NOR\_CMD\_DATA\_READ\_RESET  
NOR\_CMD\_DATA\_FIRST  
NOR\_CMD\_DATA\_SECOND  
NOR\_CMD\_DATA\_AUTO\_SELECT  
NOR\_CMD\_DATA\_PROGRAM  
NOR\_CMD\_DATA\_CHIP\_BLOCK\_ERASE\_THIRD  
NOR\_CMD\_DATA\_CHIP\_BLOCK\_ERASE\_FOURTH  
NOR\_CMD\_DATA\_CHIP\_BLOCK\_ERASE\_FIFTH  
NOR\_CMD\_DATA\_CHIP\_ERASE  
NOR\_CMD\_DATA\_CFI  
NOR\_CMD\_DATA\_BUFFER\_AND\_PROG  
NOR\_CMD\_DATA\_BUFFER\_AND\_PROG\_CONFIRM  
NOR\_CMD\_DATA\_BLOCK\_ERASE  
NOR\_MASK\_STATUS\_DQ5  
NOR\_MASK\_STATUS\_DQ6  
MC\_ADDRESS  
DEVICE\_CODE1\_ADDR  
DEVICE\_CODE2\_ADDR  
DEVICE\_CODE3\_ADDR  
CFI1\_ADDRESS  
CFI2\_ADDRESS  
CFI3\_ADDRESS  
CFI4\_ADDRESS  
NOR\_TMEOUT  
NOR\_MEMORY\_8B  
NOR\_MEMORY\_16B  
NOR\_MEMORY\_ADRESS1  
NOR\_MEMORY\_ADRESS2  
NOR\_MEMORY\_ADRESS3  
NOR\_MEMORY\_ADRESS4

**NOR Private Macros**

\_\_NOR\_ADDR\_SHIFT    **Description:**

- NOR memory address shifting.

**Parameters:**

- \_\_NOR\_ADDRESS: NOR base address



- `__NOR_MEMORY_WIDTH__`: NOR memory width
- `__ADDRESS__`: NOR memory address

**Return value:**

- NOR: shifted address value

**Description:**

- NOR memory write data to specified address.

**Parameters:**

- `__ADDRESS__`: NOR memory address
- `__DATA__`: Data to write

**Return value:**

- None:

`__NOR_WRITE`

## 27 HAL PCCARD Generic Driver

### 27.1 PCCARD Firmware driver registers structures

#### 27.1.1 PCCARD\_HandleTypeDef

*PCCARD\_HandleTypeDef* is defined in the `stm32f1xx_hal_pccard.h`

##### Data Fields

- *FSMC\_PCCARD\_TypeDef \* Instance*
- *FSMC\_PCCARD\_InitTypeDef Init*
- *\_\_IO HAL\_PCCARD\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*

##### Field Documentation

- *FSMC\_PCCARD\_TypeDef\* PCCARD\_HandleTypeDef::Instance* Register base address for PCCARD device
- *FSMC\_PCCARD\_InitTypeDef PCCARD\_HandleTypeDef::Init* PCCARD device control configuration parameters
- *\_\_IO HAL\_PCCARD\_StateTypeDef PCCARD\_HandleTypeDef::State* PCCARD device access state
- *HAL\_LockTypeDef PCCARD\_HandleTypeDef::Lock* PCCARD Lock

### 27.2 PCCARD Firmware driver API description

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

#### 27.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control PCCARD/compact flash memories. It uses the FSMC/FSMC layer functions to interface with PCCARD devices. This driver is used for:

- PCCARD/compact flash memory configuration sequence using the function `HAL_PCCARD_Init()` with control and timing parameters for both common and attribute spaces.
- Read PCCARD/compact flash memory maker and device IDs using the function `HAL_CF_Read_ID()`. The read information is stored in the `CompactFlash_ID` structure declared by the function caller.
- Access PCCARD/compact flash memory by read/write operations using the functions `HAL_CF_Read_Sector()`/`HAL_CF_Write_Sector()`, to read/write sector.
- Perform PCCARD/compact flash Reset chip operation using the function `HAL_CF_Reset()`.
- Perform PCCARD/compact flash erase sector operation using the function `HAL_CF_Erase_Sector()`.
- Read the PCCARD/compact flash status operation using the function `HAL_CF_ReadStatus()`.

- You can monitor the PCCARD/compact flash device HAL state by calling the function `HAL_PCCARD_GetState()`



This driver is a set of generic APIs which handle standard PCCARD/compact flash operations. If a PCCARD/compact flash device contains different operations and/or implementations, it should be implemented separately.

## 27.2.2 PCCARD Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the PCCARD memory

- [`HAL\_PCCARD\_Init\(\)`](#)
- [`HAL\_PCCARD\_DeInit\(\)`](#)
- [`HAL\_PCCARD\_Msplnit\(\)`](#)
- [`HAL\_PCCARD\_MspDeInit\(\)`](#)

## 27.2.3 PCCARD Input Output and memory functions

This section provides functions allowing to use and control the PCCARD memory

- [`HAL\_CF\_Read\_ID\(\)`](#)
- [`HAL\_CF\_Read\_Sector\(\)`](#)
- [`HAL\_CF\_Write\_Sector\(\)`](#)
- [`HAL\_CF\_Erase\_Sector\(\)`](#)
- [`HAL\_CF\_Reset\(\)`](#)
- [`HAL\_PCCARD\_IRQHandler\(\)`](#)
- [`HAL\_PCCARD\_ITCallback\(\)`](#)

## 27.2.4 PCCARD Peripheral State functions

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

- [`HAL\_PCCARD\_GetState\(\)`](#)
- [`HAL\_CF\_GetStatus\(\)`](#)
- [`HAL\_CF\_ReadStatus\(\)`](#)

## 27.2.5 HAL\_PCCARD\_Init

Function Name	<code>HAL_StatusTypeDef HAL_PCCARD_Init</code> ( <code>PCCARD_HandleTypeDef * hpccard</code> , <code>FSMC_NAND_PCC_TimingTypeDef * ComSpaceTiming</code> , <code>FSMC_NAND_PCC_TimingTypeDef * AttSpaceTiming</code> , <code>FSMC_NAND_PCC_TimingTypeDef * IOSpaceTiming</code> )
Function Description	Perform the PCCARD memory Initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b>: pointer to a <code>PCCARD_HandleTypeDef</code> structure that contains the configuration information for PCCARD module.</li> <li>• <b>ComSpaceTiming</b>: Common space timing structure</li> </ul>

- **AttSpaceTiming:** Attribute space timing structure
  - **IOSpaceTiming:** IO space timing structure
- Return values
- HAL status

## 27.2.6 HAL\_PCCARD\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_PCCARD_DeInit (PCCARD_HandleTypeDef * hpccard)</b>
Function Description	Perform the PCCARD memory De-initialization sequence.
Parameters	<ul style="list-style-type: none"><li>• <b>hpccard:</b> pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 27.2.7 HAL\_PCCARD\_Msplnit

Function Name	<b>void HAL_PCCARD_Msplnit (PCCARD_HandleTypeDef * hpccard)</b>
Function Description	PCCARD MSP Init.
Parameters	<ul style="list-style-type: none"><li>• <b>hpccard:</b> pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 27.2.8 HAL\_PCCARD\_MspDeInit

Function Name	<b>void HAL_PCCARD_MspDeInit (PCCARD_HandleTypeDef * hpccard)</b>
Function Description	PCCARD MSP DeInit.
Parameters	<ul style="list-style-type: none"><li>• <b>hpccard:</b> pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 27.2.9 HAL\_CF\_Read\_ID

Function Name	<b>HAL_StatusTypeDef HAL_CF_Read_ID (PCCARD_HandleTypeDef * hpccard, uint8_t CompactFlash_ID, uint8_t * pStatus)</b>
Function Description	Read Compact Flash's ID.
Parameters	<ul style="list-style-type: none"><li>• <b>hpccard:</b> pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD</li></ul>

module.

- **CompactFlash\_ID**: Compact flash ID structure.
- **pStatus**: pointer to compact flash status

Return values

- HAL status

## 27.2.10 HAL\_CF\_Read\_Sector

**Function Name** `HAL_StatusTypeDef HAL_CF_Read_Sector(PCCARD_HandleTypeDef * hpccard, uint16_t * pBuffer, uint16_t SectorAddress, uint8_t * pStatus)`

**Function Description** Read sector from PCCARD memory.

- Parameters**
- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.
  - **pBuffer**: pointer to destination read buffer
  - **SectorAddress**: Sector address to read
  - **pStatus**: pointer to CF status

Return values

- HAL status

## 27.2.11 HAL\_CF\_Write\_Sector

**Function Name** `HAL_StatusTypeDef HAL_CF_Write_Sector(PCCARD_HandleTypeDef * hpccard, uint16_t * pBuffer, uint16_t SectorAddress, uint8_t * pStatus)`

**Function Description** Write sector to PCCARD memory.

- Parameters**
- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.
  - **pBuffer**: pointer to source write buffer
  - **SectorAddress**: Sector address to write
  - **pStatus**: pointer to CF status

Return values

- HAL status

## 27.2.12 HAL\_CF\_Erase\_Sector

**Function Name** `HAL_StatusTypeDef HAL_CF_Erase_Sector(PCCARD_HandleTypeDef * hpccard, uint16_t SectorAddress, uint8_t * pStatus)`

**Function Description** Erase sector from PCCARD memory.

- Parameters**
- **hpccard**: pointer to a PCCARD\_HandleTypeDef structure that contains the configuration information for PCCARD module.
  - **SectorAddress**: Sector address to erase
  - **pStatus**: pointer to CF status

Return values

- HAL status

### 27.2.13 HAL\_CF\_Reset

Function Name **HAL\_StatusTypeDef HAL\_CF\_Reset (PCCARD\_HandleTypeDef \* hpccard)**

Function Description Reset the PCCARD memory.

Parameters

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

Return values

- HAL status

### 27.2.14 HAL\_PCCARD\_IRQHandler

Function Name **void HAL\_PCCARD\_IRQHandler (PCCARD\_HandleTypeDef \* hpccard)**

Function Description This function handles PCCARD device interrupt request.

Parameters

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

Return values

- HAL status

### 27.2.15 HAL\_PCCARD\_ITCallback

Function Name **void HAL\_PCCARD\_ITCallback (PCCARD\_HandleTypeDef \* hpccard)**

Function Description PCCARD interrupt feature callback.

Parameters

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

Return values

- None

### 27.2.16 HAL\_PCCARD\_GetState

Function Name **HAL\_PCCARD\_StateTypeDef HAL\_PCCARD\_GetState (PCCARD\_HandleTypeDef \* hpccard)**

Function Description return the PCCARD controller state

Parameters

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

Return values

- HAL state

## 27.2.17 HAL\_CF\_GetStatus

Function Name	<b>CF_StatusTypeDef HAL_CF_GetStatus (PCCARD_HandleTypeDef * hpccard)</b>
Function Description	Get the compact flash memory status.
Parameters	<ul style="list-style-type: none"> <li><b>hpccard:</b> pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>New status of the CF operation. This parameter can be: CompactFlash_TIMEOUT_ERROR: when the previous operation generate a Timeout error CompactFlash_READY: when memory is ready for the next operation</li> </ul>

## 27.2.18 HAL\_CF\_ReadStatus

Function Name	<b>CF_StatusTypeDef HAL_CF_ReadStatus (PCCARD_HandleTypeDef * hpccard)</b>
Function Description	Reads the Compact Flash memory status using the Read status command.
Parameters	<ul style="list-style-type: none"> <li><b>hpccard:</b> pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>The status of the Compact Flash memory. This parameter can be: CompactFlash_BUSY: when memory is busy CompactFlash_READY: when memory is ready for the next operation CompactFlash_ERROR: when the previous operation generates error</li> </ul>

## 27.3 PCCARD Firmware driver defines

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

### 27.3.1 PCCARD

PCCARD

#### *PCCARD Exported Macros*

<b>__HAL_PCCARD_RESET_HANDLE_STATE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset PCCARD handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__:</b> specifies the PCCARD handle.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None:</li> </ul>
--	--

#### *PCCARD Private Constants*

PCCARD\_TIMEOUT\_READ\_ID  
PCCARD\_TIMEOUT\_SECTOR  
PCCARD\_TIMEOUT\_STATUS  
PCCARD\_STATUS\_OK  
PCCARD\_STATUS\_WRITE\_OK  
CF\_DEVICE\_ADDRESS  
CF\_ATTRIBUTE\_SPACE\_ADDRESS  
CF\_COMMON\_SPACE\_ADDRESS  
CF\_IO\_SPACE\_ADDRESS  
CF\_IO\_SPACE\_PRIMARY\_ADDR  
CF\_DATA  
CF\_SECTOR\_COUNT  
CF\_SECTOR\_NUMBER  
CF\_CYLINDER\_LOW  
CF\_CYLINDER\_HIGH  
CF\_CARD\_HEAD  
CF\_STATUS\_CMD  
CF\_STATUS\_CMD\_ALTERNATE  
CF\_COMMON\_DATA\_AREA  
CF\_CARD\_CONFIGURATION  
CF\_READ\_SECTOR\_CMD  
CF\_WRITE\_SECTOR\_CMD  
CF\_ERASE\_SECTOR\_CMD  
CF\_IDENTIFY\_CMD  
CF\_TIMEOUT\_ERROR  
CF\_BUSY  
CF\_PROGR  
CF\_READY  
CF\_SECTOR\_SIZE



## 28 HAL PCD Generic Driver

### 28.1 PCD Firmware driver registers structures

#### 28.1.1 PCD\_HandleTypeDef

*PCD\_HandleTypeDef* is defined in the stm32f1xx\_hal\_pcd.h

##### Data Fields

- *PCD\_TypeDef \* Instance*
- *PCD\_InitTypeDef Init*
- *\_\_IO uint8\_t USB\_Address*
- *PCD\_EPTypeDef IN\_ep*
- *PCD\_EPTypeDef OUT\_ep*
- *HAL\_LockTypeDef Lock*
- *\_\_IO PCD\_StateTypeDef State*
- *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: not used by USB OTG FS
- *PCD\_EPTypeDef PCD\_HandleTypeDef::IN\_ep[15]* IN endpoint parameters
- *PCD\_EPTypeDef PCD\_HandleTypeDef::OUT\_ep[15]* OUT endpoint parameters
- *HAL\_LockTypeDef PCD\_HandleTypeDef::Lock* PCD peripheral status
- *\_\_IO PCD\_StateTypeDef PCD\_HandleTypeDef::State* PCD communication state
- *uint32\_t PCD\_HandleTypeDef::Setup[12]* Setup packet buffer
- *void\* PCD\_HandleTypeDef::pData* Pointer to upper stack Handler

### 28.2 PCD Firmware driver API description

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

#### 28.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD\_HandleTypeDef handle structure, for example: PCD\_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL\_PCD\_Init() API to initialize the HCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL\_PCD\_MspInit() API:
  - a. Enable the PCD/USB Low Level interface clock using the following macro
    - `__HAL_RCC_USB_CLK_ENABLE();` For USB Device FS peripheral available on STM32F102xx and STM32F103xx devices

- `__HAL_RCC_OTGFS_CLK_ENABLE();` For USB OTG FS peripheral available on STM32F105xx and STM32F107xx devices
- 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();`

## 28.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

## 28.2.3 IO operation functions

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

- [`HAL\_PCD\_Start\(\)`](#)
- [`HAL\_PCD\_Stop\(\)`](#)
- [`HAL\_PCD\_IRQHandler\(\)`](#)
- [`HAL\_PCD\_DataOutStageCallback\(\)`](#)
- [`HAL\_PCD\_DataInStageCallback\(\)`](#)
- [`HAL\_PCD\_SetupStageCallback\(\)`](#)
- [`HAL\_PCD\_SOFCallback\(\)`](#)
- [`HAL\_PCD\_ResetCallback\(\)`](#)
- [`HAL\_PCD\_SuspendCallback\(\)`](#)
- [`HAL\_PCD\_ResumeCallback\(\)`](#)
- [`HAL\_PCD\_ISOOUTIncompleteCallback\(\)`](#)
- [`HAL\_PCD\_ISOINIncompleteCallback\(\)`](#)
- [`HAL\_PCD\_ConnectCallback\(\)`](#)
- [`HAL\_PCD\_DisconnectCallback\(\)`](#)

## 28.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.

- [`HAL\_PCD\_DevConnect\(\)`](#)
- [`HAL\_PCD\_DevDisconnect\(\)`](#)
- [`HAL\_PCD\_SetAddress\(\)`](#)
- [`HAL\_PCD\_EP\_Open\(\)`](#)
- [`HAL\_PCD\_EP\_Close\(\)`](#)
- [`HAL\_PCD\_EP\_Receive\(\)`](#)
- [`HAL\_PCD\_EP\_GetRxCount\(\)`](#)
- [`HAL\_PCD\_EP\_Transmit\(\)`](#)
- [`HAL\_PCD\_EP\_SetStall\(\)`](#)
- [`HAL\_PCD\_EP\_ClrStall\(\)`](#)

- [HAL\\_PCD\\_EP\\_Flush\(\)](#)
- [HAL\\_PCD\\_ActiveRemoteWakeup\(\)](#)
- [HAL\\_PCD\\_DeActiveRemoteWakeup\(\)](#)

## 28.2.5 Peripheral State functions

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

- [HAL\\_PCD\\_GetState\(\)](#)

## 28.2.6 HAL\_PCD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)</b>
Function Description	Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 28.2.7 HAL\_PCD\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)</b>
Function Description	DeInitializes the PCD peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 28.2.8 HAL\_PCD\_MspInit

Function Name	<b>void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)</b>
Function Description	Initializes the PCD MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 28.2.9 HAL\_PCD\_MspDeInit

Function Name	<b>void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)</b>
Function Description	DeInitializes PCD MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 28.2.10 HAL\_PCD\_Start

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)</b>
Function Description	Start The USB Device.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 28.2.11 HAL\_PCD\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)</b>
Function Description	Stop The USB Device.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 28.2.12 HAL\_PCD\_IRQHandler

Function Name	<b>void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)</b>
Function Description	This function handles PCD interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 28.2.13 HAL\_PCD\_DataOutStageCallback

Function Name	<b>void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)</b>
Function Description	Data out stage callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li><li>• <b>epnum</b>: endpoint number</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 28.2.14 HAL\_PCD\_DataInStageCallback

Function Name	<b>void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)</b>
Function Description	Data IN stage callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li><li>• <b>epnum</b>: endpoint number</li></ul>

Return values

- None

## 28.2.15 HAL\_PCD\_SetupStageCallback

Function Name **void HAL\_PCD\_SetupStageCallback (PCD\_HandleTypeDef \* hpcd)**

Function Description Setup stage callback.

Parameters

- **hpcd**: PCD handle

Return values

- None

## 28.2.16 HAL\_PCD\_SOFCallback

Function Name **void HAL\_PCD\_SOFCallback (PCD\_HandleTypeDef \* hpcd)**

Function Description USB Start Of Frame callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

## 28.2.17 HAL\_PCD\_ResetCallback

Function Name **void HAL\_PCD\_ResetCallback (PCD\_HandleTypeDef \* hpcd)**

Function Description USB Reset callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

## 28.2.18 HAL\_PCD\_SuspendCallback

Function Name **void HAL\_PCD\_SuspendCallback (PCD\_HandleTypeDef \* hpcd)**

Function Description Suspend event callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

## 28.2.19 HAL\_PCD\_ResumeCallback

Function Name **void HAL\_PCD\_ResumeCallback (PCD\_HandleTypeDef \* hpcd)**

Function Description Resume event callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

## 28.2.20 HAL\_PCD\_ISOOUTIncompleteCallback

Function Name **void HAL\_PCD\_ISOOUTIncompleteCallback (PCD\_HandleTypeDef \* hpcd, uint8\_t epnum)**

Function Description Incomplete ISO OUT callbacks.

Parameters

- **hpcd**: PCD handle
- **epnum**: endpoint number

Return values

- None

## 28.2.21 HAL\_PCD\_ISOINIncompleteCallback

Function Name **void HAL\_PCD\_ISOINIncompleteCallback (PCD\_HandleTypeDef \* hpcd, uint8\_t epnum)**

Function Description Incomplete ISO IN callbacks.

Parameters

- **hpcd**: PCD handle
- **epnum**: endpoint number

Return values

- None

## 28.2.22 HAL\_PCD\_ConnectCallback

Function Name **void HAL\_PCD\_ConnectCallback (PCD\_HandleTypeDef \* hpcd)**

Function Description Connection event callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

## 28.2.23 HAL\_PCD\_DisconnectCallback

Function Name **void HAL\_PCD\_DisconnectCallback (PCD\_HandleTypeDef \* hpcd)**

Function Description Disconnection event callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

## 28.2.24 HAL\_PCD\_DevConnect

Function Name **HAL\_StatusTypeDef HAL\_PCD\_DevConnect (PCD\_HandleTypeDef \* hpcd)**

Function Description	Connect the USB device.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 28.2.25 HAL\_PCD\_DevDisconnect

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DevDisconnect</b> (PCD_HandleTypeDef * hpcd)
Function Description	Disconnect the USB device.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 28.2.26 HAL\_PCD\_SetAddress

Function Name	<b>HAL_StatusTypeDef HAL_PCD_SetAddress</b> (PCD_HandleTypeDef * hpcd, uint8_t address)
Function Description	Set the USB Device address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> <li>• <b>address</b>: new device address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 28.2.27 HAL\_PCD\_EP\_Open

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Open</b> (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)
Function Description	Open and configure an endpoint.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> <li>• <b>ep_addr</b>: endpoint address</li> <li>• <b>ep_mps</b>: endpoint max packet size</li> <li>• <b>ep_type</b>: endpoint type</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 28.2.28 HAL\_PCD\_EP\_Close

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Close</b> (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function Description	Deactivate an endpoint.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> <li>• <b>ep_addr</b>: endpoint address</li> </ul>

Return values

- HAL status

### 28.2.29 HAL\_PCD\_EP\_Receive

**Function Name** HAL\_StatusTypeDef HAL\_PCD\_EP\_Receive  
(PCD\_HandleTypeDef \* hpcd, uint8\_t ep\_addr, uint8\_t \* pBuf, uint32\_t len)

**Function Description** Receive an amount of data.

**Parameters**

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address
- **pBuf**: pointer to the reception buffer
- **len**: amount of data to be received

**Return values**

- HAL status

### 28.2.30 HAL\_PCD\_EP\_GetRxCount

**Function Name** uint16\_t HAL\_PCD\_EP\_GetRxCount (PCD\_HandleTypeDef \* hpcd, uint8\_t ep\_addr)

**Function Description** Get Received Data Size.

**Parameters**

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address

**Return values**

- Data Size

### 28.2.31 HAL\_PCD\_EP\_Transmit

**Function Name** HAL\_StatusTypeDef HAL\_PCD\_EP\_Transmit  
(PCD\_HandleTypeDef \* hpcd, uint8\_t ep\_addr, uint8\_t \* pBuf, uint32\_t len)

**Function Description** Send an amount of data.

**Parameters**

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address
- **pBuf**: pointer to the transmission buffer
- **len**: amount of data to be sent

**Return values**

- HAL status

### 28.2.32 HAL\_PCD\_EP\_SetStall

**Function Name** HAL\_StatusTypeDef HAL\_PCD\_EP\_SetStall  
(PCD\_HandleTypeDef \* hpcd, uint8\_t ep\_addr)

**Function Description** Set a STALL condition over an endpoint.

**Parameters**

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address



Return values

- HAL status

### 28.2.33 HAL\_PCD\_EP\_ClrStall

Function Name **HAL\_StatusTypeDef HAL\_PCD\_EP\_ClrStall (PCD\_HandleTypeDef \* hpcd, uint8\_t ep\_addr)**

Function Description Clear a STALL condition over in an endpoint.

Parameters

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address

Return values

- HAL status

### 28.2.34 HAL\_PCD\_EP\_Flush

Function Name **HAL\_StatusTypeDef HAL\_PCD\_EP\_Flush (PCD\_HandleTypeDef \* hpcd, uint8\_t ep\_addr)**

Function Description Flush an endpoint.

Parameters

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address

Return values

- HAL status

### 28.2.35 HAL\_PCD\_ActiveRemoteWakeup

Function Name **HAL\_StatusTypeDef HAL\_PCD\_ActiveRemoteWakeup (PCD\_HandleTypeDef \* hpcd)**

Function Description HAL\_PCD\_ActiveRemoteWakeup : active remote wakeup signalling.

Parameters

- **hpcd**: PCD handle

Return values

- HAL status

### 28.2.36 HAL\_PCD\_DeActiveRemoteWakeup

Function Name **HAL\_StatusTypeDef HAL\_PCD\_DeActiveRemoteWakeup (PCD\_HandleTypeDef \* hpcd)**

Function Description HAL\_PCD\_DeActiveRemoteWakeup : de-active remote wakeup signalling.

Parameters

- **hpcd**: PCD handle

Return values

- HAL status

### 28.2.37 HAL\_PCD\_GetState

Function Name	<b>PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)</b>
Function Description	Return the PCD state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

## 28.3 PCD Firmware driver defines

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

### 28.3.1 PCD

PCD

#### **PCD ENDP**

PCD\_ENDP0

PCD\_ENDP1

PCD\_ENDP2

PCD\_ENDP3

PCD\_ENDP4

PCD\_ENDP5

PCD\_ENDP6

PCD\_ENDP7

#### **PCD Endpoint Kind**

PCD\_SNG\_BUF

PCD\_DBL\_BUF

#### **PCD EP0 MPS**

PCD\_EP0MPS\_64

PCD\_EP0MPS\_32

PCD\_EP0MPS\_16

PCD\_EP0MPS\_08

#### **PCD Exported Macros**

\_\_HAL\_PCD\_ENABLE

\_\_HAL\_PCD\_DISABLE

\_\_HAL\_PCD\_GET\_FLAG

\_\_HAL\_PCD\_CLEAR\_FLAG

\_\_HAL\_USB\_WAKEUP\_EXTI\_ENABLE\_IT

\_\_HAL\_USB\_WAKEUP\_EXTI\_DISABLE\_IT

\_\_HAL\_USB\_WAKEUP\_EXTI\_GET\_FLAG

\_\_HAL\_USB\_WAKEUP\_EXTI\_CLEAR\_FLAG

\_\_HAL\_USB\_WAKEUP\_EXTI\_ENABLE\_RISING\_EDGE

\_\_HAL\_USB\_WAKEUP\_EXTI\_ENABLE\_FALLING\_EDGE

\_\_HAL\_USB\_WAKEUP\_EXTI\_ENABLE\_RISING\_FALLING\_EDGE

**PCD Instance definition**

IS\_PCD\_ALL\_INSTANCE

**PCD PHY Module**

PCD\_PHY\_EMBEDDED

**PCD Private Macros**

PCD\_MIN

PCD\_MAX

PCD\_SET\_ENDPOINT

PCD\_GET\_ENDPOINT

USB\_EP0StartXfer

PCD\_SET\_EPTYPE

**Description:**

- sets the type in the endpoint register(bits EP\_TYPE[1:0])

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wType: Endpoint Type.

**Return value:**

- None:

PCD\_GET\_EPTYPE

**Description:**

- gets the type in the endpoint register(bits EP\_TYPE[1:0])

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- Endpoint: Type

PCD\_FreeUserBuffer

**Description:**

- free buffer used from the application realizing it to the line toggles bit SW\_BUF in the double buffered endpoint register

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**PCD\_GET\_DB\_DIR**

- bDir: Direction

**Return value:**

- None:

**Description:**

- gets direction of the double buffered endpoint

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- EP\_DBUF\_OUT: if the endpoint counter not yet programmed.

**PCD\_SET\_EP\_TX\_STATUS****Description:**

- sets the status for tx transfer (bits STAT\_TX[1:0]).

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wState: new state

**Return value:**

- None:

**PCD\_SET\_EP\_RX\_STATUS****Description:**

- sets the status for rx transfer (bits STAT\_RX[1:0])

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wState: new state

**Return value:**

- None:

**PCD\_SET\_EP\_TXRX\_STATUS****Description:**

- sets the status for rx & tx (bits STAT\_TX[1:0] & STAT\_RX[1:0])

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wStaterx: new state.
- wStatetx: new state.

**PCD\_GET\_EP\_TX\_STATUS****Return value:**

- None:

**Description:**

- gets the status for tx/rx transfer (bits STAT\_TX[1:0] /STAT\_RX[1:0])

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- status:

**PCD\_GET\_EP\_RX\_STATUS****PCD\_SET\_EP\_TX\_VALID****Description:**

- sets directly the VALID tx/rx-status into the endpoint register

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- None:

**PCD\_SET\_EP\_RX\_VALID****PCD\_GET\_EP\_TX\_STALL\_STATUS****Description:**

- checks stall condition in an endpoint.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- TRUE: = endpoint in stall condition.

**PCD\_GET\_EP\_RX\_STALL\_STATUS****PCD\_SET\_EP\_KIND****Description:**

- set & clear EP\_KIND bit.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- None:

PCD\_CLEAR\_EP\_KIND

PCD\_SET\_OUT\_STATUS

**Description:**

- Sets/clears directly STATUS\_OUT bit in the endpoint register.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- None:

PCD\_CLEAR\_OUT\_STATUS

PCD\_SET\_EP\_DBUF

**Description:**

- Sets/clears directly EP\_KIND bit in the endpoint register.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- None:

PCD\_CLEAR\_EP\_DBUF

PCD\_CLEAR\_RX\_EP\_CTR

**Description:**

- Clears bit CTR\_RX / CTR\_TX in the endpoint register.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- None:

PCD\_CLEAR\_TX\_EP\_CTR

PCD\_RX\_DTOG

**Description:**

- Toggles DTOG\_RX / DTOG\_TX bit in the endpoint register.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- None:

PCD\_TX\_DTOG

PCD\_CLEAR\_RX\_DTOG

**Description:**

- Clears DTOG\_RX / DTOG\_TX bit in the endpoint register.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- None:

PCD\_CLEAR\_TX\_DTOG

PCD\_SET\_EP\_ADDRESS

**Description:**

- Sets address in an endpoint register.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- bAddr: Address.

**Return value:**

- None:

PCD\_GET\_EP\_ADDRESS

PCD\_EP\_TX\_ADDRESS

PCD\_EP\_TX\_CNT

PCD\_EP\_RX\_ADDRESS

PCD\_EP\_RX\_CNT

PCD\_SET\_EP\_RX\_CNT

PCD\_SET\_EP\_TX\_ADDRESS

**Description:**

- sets address of the tx/rx buffer.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wAddr: address to be set (must be word aligned).

**Return value:**

- None:

PCD\_SET\_EP\_RX\_ADDRESS

PCD\_GET\_EP\_TX\_ADDRESS

**Description:**

- Gets address of the tx/rx buffer.

PCD\_GET\_EP\_RX\_ADDRESS

PCD\_CALC\_BLK32

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- address: of the buffer.

**Description:**

- Sets counter of rx buffer with no.

**Parameters:**

- dwReg: Register
- wCount: Counter.
- wNBlocks: no. of Blocks.

**Return value:**

- None:

PCD\_CALC\_BLK2

PCD\_SET\_EP\_CNT\_RX\_REG

PCD\_SET\_EP\_RX\_DBUF0\_CNT

PCD\_SET\_EP\_TX\_CNT

**Description:**

- sets counter for the tx/rx buffer.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wCount: Counter value.

**Return value:**

- None:

PCD\_GET\_EP\_TX\_CNT

**Description:**

- gets counter of the tx buffer.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- Counter: value

PCD\_GET\_EP\_RX\_CNT

PCD\_SET\_EP\_DBUF0\_ADDR

**Description:**

- Sets buffer 0/1 address in a double buffer endpoint.



PCD\_SET\_EP\_DBUF1\_ADDR  
PCD\_SET\_EP\_DBUF\_ADDR

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.

**Return value:**

- Counter: value

**Description:**

- Sets addresses in a double buffer endpoint.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.
- wBuf1Addr: = buffer 1 address.

**Return value:**

- None:

PCD\_GET\_EP\_DBUF0\_ADDR

**Description:**

- Gets buffer 0/1 address of a double buffer endpoint.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- None:

PCD\_GET\_EP\_DBUF1\_ADDR  
PCD\_SET\_EP\_DBUF0\_CNT

**Description:**

- Gets buffer 0/1 address of a double buffer endpoint.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- bDir: endpoint dir EP\_DBUF\_OUT = OUT  
EP\_DBUF\_IN = IN
- wCount: Counter value

**Return value:**

- None:

PCD\_SET\_EP\_DBUF1\_CNT

PCD\_SET\_EP\_DBUF\_CNT

PCD\_GET\_EP\_DBUF0\_CNT

**Description:**

- Gets buffer 0/1 rx/tx counter for double buffering.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- None:

PCD\_GET\_EP\_DBUF1\_CNT

***PCD Speed***

PCD\_SPEED\_HIGH

PCD\_SPEED\_HIGH\_IN\_FULL

PCD\_SPEED\_FULL

## 29 HAL PCD Extension Driver

### 29.1 PCDEx Firmware driver API description

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

#### 29.1.1 Extended Peripheral Control functions

This section provides functions allowing to:

- Update FIFO (USB\_OTG\_FS)
- Update PMA configuration (USB)
- [HAL\\_PCDEx\\_PMAConfig\(\)](#)

#### 29.1.2 HAL\_PCDEx\_PMAConfig

Function Name	<b>HAL_StatusTypeDef HAL_PCDEx_PMAConfig</b> (PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t ep_kind, uint32_t pmaaddress)
Function Description	Configure PMA for EP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: : Device instance</li> <li>• <b>ep_addr</b>: endpoint address</li> <li>• <b>ep_kind</b>: endpoint Kind USB_SNG_BUF: Single Buffer used USB_DBL_BUF: Double Buffer used</li> <li>• <b>pmaaddress</b>: 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.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 29.1.3 HAL\_PCDEx\_SetConnectionState

Function Name	<b>void HAL_PCDEx_SetConnectionState</b> (PCD_HandleTypeDef * hpcd, uint8_t state)
Function Description	Software Device Connection, this function is not required by USB OTG FS peripheral, it is used only by USB Device FS peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> <li>• <b>state</b>: connection state (0 : disconnected / 1: connected)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 29.2 PCDEx Firmware driver defines

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

---

## 29.2.1 PCDEx

PCDEx

## 30 HAL PWR Generic Driver

### 30.1 PWR Firmware driver registers structures

#### 30.1.1 PWR\_PVDTypeDef

*PWR\_PVDTypeDef* is defined in the stm32f1xx\_hal\_pwr.h

##### Data Fields

- *uint32\_t PVDLevel*
- *uint32\_t Mode*

##### Field Documentation

- *uint32\_t PWR\_PVDTypeDef::PVDLevel* PVDLevel: Specifies the PVD detection level. This parameter can be a value of [PWR\\_PVD\\_detection\\_level](#)
- *uint32\_t PWR\_PVDTypeDef::Mode* Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [PWR\\_PVD\\_Mode](#)

### 30.2 PWR Firmware driver API description

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

#### 30.2.1 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 `__HAL_RCC_PWR_CLK_ENABLE()` macro.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.
- [HAL\\_PWR\\_DeInit\(\)](#)
- [HAL\\_PWR\\_EnableBkUpAccess\(\)](#)
- [HAL\\_PWR\\_DisableBkUpAccess\(\)](#)

#### 30.2.2 Peripheral Control functions

##### PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in the PWR\_CR).
- A PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through `__HAL_PVD_EXTI_ENABLE_IT()` macro.
- The PVD is stopped in Standby mode.

## WakeUp pin configuration

- WakeUp pin is used to wake up the system from Standby mode. This pin is forced in input pull-down configuration and is active on rising edges.
- There is one WakeUp pin: WakeUp Pin 1 on PA.00.

## Low Power modes configuration

The device features 3 low-power modes:

- Sleep mode: CPU clock off, all peripherals including Cortex-M3 core peripherals like NVIC, SysTick, etc. are kept running
- Stop mode: All clocks are stopped
- Standby mode: 1.8V domain powered off

### 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:
  - WFI entry mode, Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.
  - WFE entry mode, Any wakeup event can wake up the device from Sleep mode.
    - Any peripheral interrupt w/o NVIC configuration & SEVONPEND bit set in the Cortex (`HAL_PWR_EnableSEVOnPend`)
    - Any EXTI Line (Internal or External) configured in Event mode

### Stop mode

The Stop mode is based on the Cortex-M3 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 1.8 V domain are stopped, the PLL, the HSI and the HSE RC oscillators are disabled. SRAM and register contents are preserved. 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_REGULATOR_VALUE, PWR_SLEEPENTRY_WFx)` function with:
  - `PWR_REGULATOR_VALUE= PWR_MAINREGULATOR_ON`: Main regulator ON.
  - `PWR_REGULATOR_VALUE= PWR_LOWPOWERREGULATOR_ON`: Low Power regulator ON.
  - `PWR_SLEEPENTRY_WFx= PWR_SLEEPENTRY_WFI`: enter STOP mode with WFI instruction
  - `PWR_SLEEPENTRY_WFx= PWR_SLEEPENTRY_WFE`: enter STOP mode with WFE instruction
- Exit:
  - WFI entry mode, Any EXTI Line (Internal or External) configured in Interrupt mode with NVIC configured

- WFE entry mode, Any EXTI Line (Internal or External) configured in Event mode.

### Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M3 deepsleep mode, with the voltage regulator disabled. The 1.8 V domain is consequently powered off. The PLL, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for registers in the Backup domain and Standby circuitry

- Entry:
  - The Standby mode is entered using the HAL\_PWR\_EnterSTANDBYMode() function.
- Exit:
  - WKUP pin rising edge, RTC alarm event rising edge, external Reset in NRSTpin, IWDG Reset

### Auto-wakeup (AWU) from low-power mode

- The MCU can be woken up from low-power mode by an RTC Alarm event, without depending on an external interrupt (Auto-wakeup mode).
- RTC auto-wakeup (AWU) from the Stop and Standby modes
  - To wake up from the Stop mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL\_RTC\_SetAlarm\_IT() function.

### PWR Workarounds linked to Silicon Limitation

Below the list of all silicon limitations known on STM32F1xx product.

1. Workarounds Implemented inside PWR HAL Driver
  - a. Debugging Stop mode with WFE entry - overloaded the WFE by an internal function
    - [HAL\\_PWR\\_ConfigPVD\(\)](#)
    - [HAL\\_PWR\\_EnablePVD\(\)](#)
    - [HAL\\_PWR\\_DisablePVD\(\)](#)
    - [HAL\\_PWR\\_EnableWakeUpPin\(\)](#)
    - [HAL\\_PWR\\_DisableWakeUpPin\(\)](#)
    - [HAL\\_PWR\\_EnterSLEEPMode\(\)](#)
    - [HAL\\_PWR\\_EnterSTOPMode\(\)](#)
    - [HAL\\_PWR\\_EnterSTANDBYMode\(\)](#)
    - [HAL\\_PWR\\_EnableSleepOnExit\(\)](#)
    - [HAL\\_PWR\\_DisableSleepOnExit\(\)](#)
    - [HAL\\_PWR\\_EnableSEVOnPend\(\)](#)
    - [HAL\\_PWR\\_DisableSEVOnPend\(\)](#)
    - [HAL\\_PWR\\_PVD\\_IRQHandler\(\)](#)
    - [HAL\\_PWR\\_PVDCallback\(\)](#)

### 30.2.3 HAL\_PWR\_DeInit

Function Name	<b>void HAL_PWR_DeInit (void )</b>
Function Description	Deinitializes the PWR peripheral registers to their default reset values.

Return values

- None

### 30.2.4 HAL\_PWR\_EnableBkUpAccess

Function Name **void HAL\_PWR\_EnableBkUpAccess (void )**

Function Description Enables access to the backup domain (RTC registers, RTC backup data registers ).

Return values

- None

Notes

- If the HSE divided by 128 is used as the RTC clock, the Backup Domain Access should be kept enabled.

### 30.2.5 HAL\_PWR\_DisableBkUpAccess

Function Name **void HAL\_PWR\_DisableBkUpAccess (void )**

Function Description Disables access to the backup domain (RTC registers, RTC backup data registers).

Return values

- None

Notes

- If the HSE divided by 128 is used as the RTC clock, the Backup Domain Access should be kept enabled.

### 30.2.6 HAL\_PWR\_ConfigPVD

Function Name **void HAL\_PWR\_ConfigPVD (PWR\_PVDTypeDef \* sConfigPVD)**

Function Description Configures the voltage threshold detected by the Power Voltage Detector(PVD).

Parameters

- **sConfigPVD**: pointer to an PWR\_PVDTypeDef structure that contains the configuration information for the PVD.

Return values

- None

Notes

- Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.

### 30.2.7 HAL\_PWR\_EnablePVD

Function Name **void HAL\_PWR\_EnablePVD (void )**

Function Description Enables the Power Voltage Detector(PVD).

Return values

- None

### 30.2.8 HAL\_PWR\_DisablePVD



Function Name	<b>void HAL_PWR_DisablePVD (void )</b>
Function Description	Disables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 30.2.9 HAL\_PWR\_EnableWakeUpPin

Function Name	<b>void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)</b>
Function Description	Enables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> <li>• <b>WakeUpPinx:</b> Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values: PWR_WAKEUP_PIN1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 30.2.10 HAL\_PWR\_DisableWakeUpPin

Function Name	<b>void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)</b>
Function Description	Disables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> <li>• <b>WakeUpPinx:</b> Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values: PWR_WAKEUP_PIN1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 30.2.11 HAL\_PWR\_EnterSLEEPMode

Function Name	<b>void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)</b>
Function Description	Enters Sleep mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Regulator:</b> Regulator state as no effect in SLEEP mode - allows to support portability from legacy software</li> <li>• <b>SLEEPEntry:</b> 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</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In Sleep mode, all I/O pins keep the same state as in Run mode.</li> </ul>

### 30.2.12 HAL\_PWR\_EnterSTOPMode

Function Name	<b>void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)</b>
Function Description	Enters Stop mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Regulator:</b> Specifies the regulator state in Stop mode. This parameter can be one of the following values: PWR_MAINREGULATOR_ON: Stop mode with regulator ON PWR_LOWPOWERREGULATOR_ON: Stop mode with low power regulator ON</li> <li>• <b>STOPEntry:</b> Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values: PWR_STOPENTRY_WFI: Enter Stop mode with WFI instruction PWR_STOPENTRY_WFE: Enter Stop mode with WFE instruction</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In Stop mode, all I/O pins keep the same state as in Run mode.</li> <li>• When exiting Stop mode by using an interrupt or a wakeup event, HSI RC oscillator is selected as system clock.</li> <li>• 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.</li> </ul>

### 30.2.13 HAL\_PWR\_EnterSTANDBYMode

Function Name	<b>void HAL_PWR_EnterSTANDBYMode (void )</b>
Function Description	Enters Standby mode.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In Standby mode, all I/O pins are high impedance except for: Reset pad (still available) TAMPER pin if configured for tamper or calibration out. WKUP pin (PA0) if enabled.</li> </ul>

### 30.2.14 HAL\_PWR\_EnableSleepOnExit

Function Name	<b>void HAL_PWR_EnableSleepOnExit (void )</b>
Function Description	Indicates Sleep-On-Exit when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.</li> </ul>

### 30.2.15 HAL\_PWR\_DisableSleepOnExit

Function Name	<b>void HAL_PWR_DisableSleepOnExit (void )</b>
Function Description	Disables Sleep-On-Exit feature when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none"><li>• None</li></ul>
Notes	<ul style="list-style-type: none"><li>• Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.</li></ul>

### 30.2.16 HAL\_PWR\_EnableSEVOnPend

Function Name	<b>void HAL_PWR_EnableSEVOnPend (void )</b>
Function Description	Enables CORTEX M3 SEVONPEND bit.
Return values	<ul style="list-style-type: none"><li>• None</li></ul>
Notes	<ul style="list-style-type: none"><li>• Sets SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.</li></ul>

### 30.2.17 HAL\_PWR\_DisableSEVOnPend

Function Name	<b>void HAL_PWR_DisableSEVOnPend (void )</b>
Function Description	Disables CORTEX M3 SEVONPEND bit.
Return values	<ul style="list-style-type: none"><li>• None</li></ul>
Notes	<ul style="list-style-type: none"><li>• Clears SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.</li></ul>

### 30.2.18 HAL\_PWR\_PVD\_IRQHandler

Function Name	<b>void HAL_PWR_PVD_IRQHandler (void )</b>
Function Description	This function handles the PWR PVD interrupt request.
Return values	<ul style="list-style-type: none"><li>• None</li></ul>
Notes	<ul style="list-style-type: none"><li>• This API should be called under the PVD_IRQHandler().</li></ul>

### 30.2.19 HAL\_PWR\_PVDCallback

Function Name	<b>void HAL_PWR_PVDCallback (void )</b>
Function Description	PWR PVD interrupt callback.

Return values

- None

## 30.3 PWR Firmware driver defines

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

### 30.3.1 PWR

PWR

**PWR CR Register alias address**

LPSDSR\_BIT\_NUMBER

CR\_LPSDSR\_BB

DBP\_BIT\_NUMBER

CR\_DBP\_BB

PVDE\_BIT\_NUMBER

CR\_PVDE\_BB

**PWR CSR Register alias address**

CSR\_EWUP\_BB

**PWR Exported Macros**

\_\_HAL\_PWR\_GET\_FLAG

#### Description:

- Check PWR flag is set or not.

#### Parameters:

- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `PWR_FLAG_WU`: Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm. An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.
  - `PWR_FLAG_SB`: StandBy flag. This flag indicates that the system was resumed from StandBy mode.
  - `PWR_FLAG_PVDO`: PVD Output. This flag

is valid only if PVD is enabled by the HAL\_PWR\_EnablePVD() function. The PVD is stopped by Standby mode. For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set.

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

**Description:**

- Clear the PWR's pending flags.

**Parameters:**

- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be one of the following values:
  - PWR\_FLAG\_WU: Wake Up flag
  - PWR\_FLAG\_SB: StandBy flag

**Description:**

- Enable interrupt on PVD Exti Line 16.

**Return value:**

- None.:

**Description:**

- Disable interrupt on PVD Exti Line 16.

**Return value:**

- None.:

**Description:**

- Enable event on PVD Exti Line 16.

**Return value:**

- None.:

**Description:**

- Disable event on PVD Exti Line 16.

\_\_HAL\_PWR\_CLEAR\_FLAG

\_\_HAL\_PWR\_PVD\_EXTI\_ENABLE\_IT

\_\_HAL\_PWR\_PVD\_EXTI\_DISABLE\_IT

\_\_HAL\_PWR\_PVD\_EXTI\_ENABLE\_EVENT

\_\_HAL\_PWR\_PVD\_EXTI\_DISABLE\_EVENT

\_\_HAL\_PWR\_PVD\_EXTI\_ENABLE\_FALLING\_EDGE

**Return value:**

- None.:

**Description:**

- PVD EXTI line configuration: set falling edge trigger.

**Return value:**

- None.:

\_\_HAL\_PWR\_PVD\_EXTI\_DISABLE\_FALLING\_EDGE

**Description:**

- Disable the PVD Extended Interrupt Falling Trigger.

**Return value:**

- None.:

\_\_HAL\_PWR\_PVD\_EXTI\_ENABLE\_RISING\_EDGE

**Description:**

- PVD EXTI line configuration: set rising edge trigger.

**Return value:**

- None.:

\_\_HAL\_PWR\_PVD\_EXTI\_DISABLE\_RISING\_EDGE

**Description:**

- Disable the PVD Extended Interrupt Rising Trigger.

**Return value:**

- None.:

\_\_HAL\_PWR\_PVD\_EXTI\_ENABLE\_RISING\_FALLING\_EDGE

**Description:**

- PVD EXTI line configuration: set rising & falling edge trigger.

**Return value:**

- None.:

\_\_HAL\_PWR\_PVD\_EXTI\_DISABLE\_RISING\_FALLING\_EDGE

**Description:**

- Disable the PVD Extended Interrupt Rising & Falling Trigger.

**Return value:**

- None.:

\_\_HAL\_PWR\_PVD\_EXTI\_GET\_FLAG

**Description:**

- Check whether the specified PVD EXTI interrupt flag is set or not.

\_\_HAL\_PWR\_PVD\_EXTI\_CLEAR\_FLAG

**Return value:**

- EXTI: PVD Line Status.

**Description:**

- Clear the PVD EXTI flag.

**Return value:**

- None.:

**Description:**

- Generate a Software interrupt on selected EXTI line.

**Return value:**

- None.:

\_\_HAL\_PWR\_PVD\_EXTI\_GENERATE\_SWIT

**PWR Flag**

PWR\_FLAG\_WU

PWR\_FLAG\_SB

PWR\_FLAG\_PVDO

**PWR Private Constants**

PWR\_EXTI\_LINE\_PVD External interrupt line 16 Connected to the PVD EXTI Line

**PWR Private Macros**

IS\_PWR\_PVD\_LEVEL

IS\_PWR\_PVD\_MODE

IS\_PWR\_WAKEUP\_PIN

IS\_PWR\_REGULATOR

IS\_PWR\_SLEEP\_ENTRY

IS\_PWR\_STOP\_ENTRY

**PWR PVD detection level**

PWR\_PVDLEVEL\_0

PWR\_PVDLEVEL\_1

PWR\_PVDLEVEL\_2

PWR\_PVDLEVEL\_3

PWR\_PVDLEVEL\_4

PWR\_PVDLEVEL\_5

PWR\_PVDLEVEL\_6

PWR\_PVDLEVEL\_7

**PWR PVD Mode**

PWR\_PVD\_MODE\_NORMAL

basic mode is used

PWR\_PVD\_MODE\_IT\_RISING

External Interrupt Mode with Rising edge

	trigger detection
PWR_PVD_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
PWR_PVD_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING	Event Mode with Rising edge trigger detection
PWR_PVD_MODE_EVENT_FALLING	Event Mode with Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING_FALLING	Event Mode with Rising/Falling edge trigger detection

**PWR PVD Mode Mask**

PVD\_MODE\_IT  
PVD\_MODE\_EVT  
PVD\_RISING\_EDGE  
PVD\_FALLING\_EDGE

**PWR Register alias address**

PWR\_OFFSET  
PWR\_CR\_OFFSET  
PWR\_CSR\_OFFSET  
PWR\_CR\_OFFSET\_BB  
PWR\_CSR\_OFFSET\_BB

**PWR Regulator state in SLEEP/STOP mode**

PWR\_MAINREGULATOR\_ON  
PWR\_LOWPOWERREGULATOR\_ON

**PWR SLEEP mode entry**

PWR\_SLEEPENTRY\_WFI  
PWR\_SLEEPENTRY\_WFE

**PWR STOP mode entry**

PWR\_STOPENTRY\_WFI  
PWR\_STOPENTRY\_WFE

**PWR WakeUp Pins**

PWR\_WAKEUP\_PIN1



## 31 HAL RCC Generic Driver

### 31.1 RCC Firmware driver registers structures

#### 31.1.1 RCC\_PLLInitTypeDef

*RCC\_PLLInitTypeDef* is defined in the stm32f1xx\_hal\_rcc.h

##### Data Fields

- *uint32\_t PLLState*
- *uint32\_t PLLSource*
- *uint32\_t PLLMUL*

##### 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* 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 input clock This parameter must be a value of [RCCEX\\_PLL\\_Multiplication\\_Factor](#)

#### 31.1.2 RCC\_ClkInitTypeDef

*RCC\_ClkInitTypeDef* is defined in the stm32f1xx\_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 RCC Firmware driver API description

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

### 31.2.1 RCC specific features

After reset the device is running from Internal High Speed oscillator (HSI 8MHz) with Flash 0 wait state, Flash prefetch buffer is enabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHB) and Low speed (APB) busses; all peripherals mapped on these busses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in input floating state, except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB busses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals whose clocks are not derived from the System clock (I2S, RTC, ADC, USB OTG FS)

### 31.2.2 RCC Limitations

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

- This delay depends on the peripheral mapping.
  - AHB & APB peripherals, 1 dummy read is necessary

Workarounds:

1. For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each `__HAL_RCC_PPP_CLK_ENABLE()` macro.

### 31.2.3 Initialization and de-initialization functions

This section provide functions allowing to configure the internal/external oscillators (HSE, HSI, LSE, LSI, PLL, CSS and MCO) and the System busses clocks (SYSCLK, AHB, APB1 and APB2).

Internal/external clock and PLL configuration

1. HSI (high-speed internal), 8 MHz factory-trimmed RC used directly or through the PLL as System clock source.
2. LSI (low-speed internal), 40 KHz low consumption RC used as IWDG and/or RTC clock source.
3. HSE (high-speed external), 4 to 24 MHz (STM32F100xx) or 4 to 16 MHz (STM32F101x/STM32F102x/STM32F103x) or 3 to 25 MHz (STM32F105x/STM32F107x) crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
4. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.

5. PLL (clocked by HSI or HSE), featuring two different output clocks:
  - The first output is used to generate the high speed system clock (up to 72 MHz for STM32F10xxx or up to 24 MHz for STM32F100xx)
  - The second output is used to generate the clock for the USB OTG FS (48 MHz)
6. CSS (Clock security system), once enable using the macro `__HAL_RCC_CSS_ENABLE()` and if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.
7. MCO1 (microcontroller clock output), used to output SYSCLK, HSI, HSE or PLL clock (divided by 2) on PA8 pin + PLL2CLK, PLL3CLK/2, PLL3CLK and XT1 for STM32F105x/STM32F107x

System, AHB and APB busses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): HSI, HSE and PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...). APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "`HAL_RCC_GetSysClockFreq()`" function to retrieve the frequencies of these clocks. All the peripheral clocks are derived from the System clock (SYSCLK) except: RTC: RTC clock can be derived either from the LSI, LSE or HSE clock divided by 128. USB OTG FS and RTC: USB OTG FS require a frequency equal to 48 MHz to work correctly. This clock is derived of the main PLL through PLL Multiplier. I2S interface on STM32F105x/STM32F107x can be derived from PLL3CLK IWDG clock which is always the LSI clock.
2. For STM32F10xxx, the maximum frequency of the SYSCLK and HCLK/PCLK2 is 72 MHz, PCLK1 36 MHz. For STM32F100xx, the maximum frequency of the SYSCLK and HCLK/PCLK1/PCLK2 is 24 MHz. Depending on the SYSCLK frequency, the flash latency should be adapted (see [Table 18: "Number of wait states \(WS\) vs SYSCLK frequency"](#)).

**Table 18: Number of wait states (WS) vs SYSCLK frequency**

Latency	SYSCLK clock frequency (MHz)
0 WS (1 CPU cycle)	$0 < \text{SYSCLK} \leq 24$
1 WS (2 CPU cycles)	$24 < \text{SYSCLK} \leq 48$
2 WS (3 CPU cycles)	$48 < \text{SYSCLK} \leq 72$

- [HAL\\_RCC\\_DeInit\(\)](#)
- [HAL\\_RCC\\_OscConfig\(\)](#)
- [HAL\\_RCC\\_ClockConfig\(\)](#)

### 31.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

- [HAL\\_RCC\\_MCOConfig\(\)](#)
- [HAL\\_RCC\\_EnableCSS\(\)](#)
- [HAL\\_RCC\\_DisableCSS\(\)](#)
- [HAL\\_RCC\\_GetSysClockFreq\(\)](#)
- [HAL\\_RCC\\_GetHCLKFreq\(\)](#)

- [HAL\\_RCC\\_GetPCLK1Freq\(\)](#)
- [HAL\\_RCC\\_GetPCLK2Freq\(\)](#)
- [HAL\\_RCC\\_GetOscConfig\(\)](#)
- [HAL\\_RCC\\_GetClockConfig\(\)](#)
- [HAL\\_RCC\\_NMI\\_IRQHandler\(\)](#)
- [HAL\\_RCC\\_CSSCallback\(\)](#)

### 31.2.5 HAL\_RCC\_DeInit

Function Name	<b>void HAL_RCC_DeInit (void )</b>
Function Description	Resets the RCC clock configuration to the default reset state.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE and PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS and MCO1 OFFAll interrupts disabled</li> <li>• This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks</li> </ul>

### 31.2.6 HAL\_RCC\_OscConfig

Function Name	<b>HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)</b>
Function Description	Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_OscInitStruct:</b> pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The PLL is not disabled when used as system clock.</li> <li>• The PLL is not disabled when USB OTG FS clock is enabled (specific to devices with USB FS)</li> </ul>

### 31.2.7 HAL\_RCC\_ClockConfig

Function Name	<b>HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)</b>
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"> <li>• <b>RCC_ClkInitStruct:</b> pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral.</li> <li>• <b>FLatency:</b> FLASH Latency This parameter can be one of the following values: FLASH_LATENCY_0: FLASH 0 Latency cycle FLASH_LATENCY_1: FLASH 1 Latency cycle FLASH_LATENCY_2: FLASH 2 Latency cycle</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>RCC_ClkInitStruct:</b> pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral.</li> <li>• <b>FLatency:</b> FLASH Latency This parameter can be one of the following values: FLASH_LATENCY_0: FLASH 0 Latency cycle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function</li> <li>• The HSI is used (enabled by hardware) as system clock source after startup from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).</li> <li>• 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. You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source.</li> <li>• The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function</li> <li>• The HSI is used (enabled by hardware) as system clock source after startup from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).</li> <li>• 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. You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source.</li> </ul>

### 31.2.8 HAL\_RCC\_MCOConfig

Function Name	<b>void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)</b>
Function Description	Selects the clock source to output on MCO pin.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_MCOx:</b> specifies the output direction for the clock source. This parameter can be one of the following values: RCC_MCO: Clock source to output on MCO1 pin(PA8).</li> <li>• <b>RCC_MCOSource:</b> specifies the clock source to output. This parameter can be one of the following values: RCC_MCO1SOURCE_NOCLOCK: No clock selected RCC_MCO1SOURCE_SYSCLK: System clock selected as MCO source RCC_MCO1SOURCE_HSI: HSI oscillator clock selected RCC_MCO1SOURCE_HSE: HSE oscillator clock</li> </ul>

selected RCC\_MCO1SOURCE\_PLLCLK: PLL clock divided by 2 selected as MCO source  
 RCC\_MCO1SOURCE\_PLL2CLK: PLL2 clock selected as MCO source (only for connectivity line devices)  
 RCC\_MCO1SOURCE\_PLL3CLK\_DIV2: PLL3 clock divided by 2 selected as MCO source (only for connectivity line devices)  
 RCC\_MCO1SOURCE\_EXT\_HSE: XT1 external 3-25 MHz oscillator clock selected as MCO source (only for connectivity line devices)  
 RCC\_MCO1SOURCE\_PLL3CLK: PLL3 clock selected as MCO source (only for connectivity line devices)

- **RCC\_MCODiv**: specifies the MCO DIV. This parameter can be one of the following values: RCC\_MCODIV\_1: no division applied to MCO clock

Return values

- None

Notes

- MCO pin should be configured in alternate function mode.

### 31.2.9 HAL\_RCC\_EnableCSS

Function Name

**void HAL\_RCC\_EnableCSS (void )**

Function Description

Enables the Clock Security System.

Return values

- None

Notes

- If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.

### 31.2.10 HAL\_RCC\_DisableCSS

Function Name

**void HAL\_RCC\_DisableCSS (void )**

Function Description

Disables the Clock Security System.

Return values

- None

### 31.2.11 HAL\_RCC\_GetSysClockFreq

Function Name

**uint32\_t HAL\_RCC\_GetSysClockFreq (void )**

Function Description

Returns the SYSCLK frequency.

Return values

- SYSCLK frequency

Notes

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- If SYSCLK source is HSI, function returns values based on HSI\_VALUE(\*)

- If SYSCLK source is HSE, function returns values based on HSE\_VALUE divided by PREDIV factor(\*\*)
- If SYSCLK source is PLL, function returns values based on HSE\_VALUE divided by PREDIV factor(\*\*) or HSI\_VALUE(\*) multiplied by the PLL factor.
- (\*) HSI\_VALUE is a constant defined in stm32f1xx\_hal\_conf.h file (default value 8 MHz).
- (\*\*) HSE\_VALUE is a constant defined in stm32f1xx\_hal\_conf.h file (default value 8 MHz), user has to ensure that HSE\_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.
- Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

### 31.2.12 HAL\_RCC\_GetHCLKFreq

Function Name	<b>uint32_t HAL_RCC_GetHCLKFreq (void )</b>
Function Description	Returns the HCLK frequency.
Return values	<ul style="list-style-type: none"> <li>• HCLK frequency</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function</li> </ul>

### 31.2.13 HAL\_RCC\_GetPCLK1Freq

Function Name	<b>uint32_t HAL_RCC_GetPCLK1Freq (void )</b>
Function Description	Returns the PCLK1 frequency.
Return values	<ul style="list-style-type: none"> <li>• PCLK1 frequency</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>

### 31.2.14 HAL\_RCC\_GetPCLK2Freq

Function Name	<b>uint32_t HAL_RCC_GetPCLK2Freq (void )</b>
Function Description	Returns the PCLK2 frequency.
Return values	<ul style="list-style-type: none"> <li>• PCLK2 frequency</li> </ul>

## Notes

- Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

### 31.2.15 HAL\_RCC\_GetOscConfig

Function Name	<b>void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)</b>
Function Description	Configures the <code>RCC_OscInitStruct</code> according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_OscInitStruct:</b> pointer to an <code>RCC_OscInitTypeDef</code> structure that will be configured.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 31.2.16 HAL\_RCC\_GetClockConfig

Function Name	<b>void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)</b>
Function Description	Configures the <code>RCC_ClkInitStruct</code> according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_ClkInitStruct:</b> pointer to an <code>RCC_ClkInitTypeDef</code> structure that will be configured.</li> <li>• <b>pFLatency:</b> Pointer on the Flash Latency.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 31.2.17 HAL\_RCC\_NMI\_IRQHandler

Function Name	<b>void HAL_RCC_NMI_IRQHandler (void )</b>
Function Description	This function handles the RCC CSS interrupt request.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API should be called under the <code>NMI_Handler()</code>.</li> </ul>

### 31.2.18 HAL\_RCC\_CSSCallback

Function Name	<b>void HAL_RCC_CSSCallback (void )</b>
Function Description	RCC Clock Security System interrupt callback.
Return values	<ul style="list-style-type: none"> <li>• none</li> </ul>

## 31.3 RCC Firmware driver defines

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



### 31.3.1 RCC

#### RCC

##### ***AHB Clock Source***

RCC_SYSCLK_DIV1	SYSCLK not divided
RCC_SYSCLK_DIV2	SYSCLK divided by 2
RCC_SYSCLK_DIV4	SYSCLK divided by 4
RCC_SYSCLK_DIV8	SYSCLK divided by 8
RCC_SYSCLK_DIV16	SYSCLK divided by 16
RCC_SYSCLK_DIV64	SYSCLK divided by 64
RCC_SYSCLK_DIV128	SYSCLK divided by 128
RCC_SYSCLK_DIV256	SYSCLK divided by 256
RCC_SYSCLK_DIV512	SYSCLK divided by 512

##### ***AHB Peripheral Clock Enable Disable Status***

\_\_HAL\_RCC\_DMA1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_DMA1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SRAM\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SRAM\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_FLITF\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_FLITF\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_CRC\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_CRC\_IS\_CLK\_DISABLED

##### ***Alias define maintained for legacy***

\_\_HAL\_RCC\_SYSCFG\_CLK\_DISABLE  
\_\_HAL\_RCC\_SYSCFG\_CLK\_ENABLE  
\_\_HAL\_RCC\_SYSCFG\_FORCE\_RESET  
\_\_HAL\_RCC\_SYSCFG\_RELEASE\_RESET

##### ***APB1 APB2 Clock Source***

RCC_HCLK_DIV1	HCLK not divided
RCC_HCLK_DIV2	HCLK divided by 2
RCC_HCLK_DIV4	HCLK divided by 4
RCC_HCLK_DIV8	HCLK divided by 8
RCC_HCLK_DIV16	HCLK divided by 16

##### ***APB1 Clock Enable Disable***

\_\_HAL\_RCC\_TIM2\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM3\_CLK\_ENABLE  
\_\_HAL\_RCC\_WWDG\_CLK\_ENABLE

\_\_HAL\_RCC\_USART2\_CLK\_ENABLE  
\_\_HAL\_RCC\_I2C1\_CLK\_ENABLE  
\_\_HAL\_RCC\_BKP\_CLK\_ENABLE  
\_\_HAL\_RCC\_PWR\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM2\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM3\_CLK\_DISABLE  
\_\_HAL\_RCC\_WWDG\_CLK\_DISABLE  
\_\_HAL\_RCC\_USART2\_CLK\_DISABLE  
\_\_HAL\_RCC\_I2C1\_CLK\_DISABLE  
\_\_HAL\_RCC\_BKP\_CLK\_DISABLE  
\_\_HAL\_RCC\_PWR\_CLK\_DISABLE

**APB1 Force Release Reset**

\_\_HAL\_RCC\_APB1\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM2\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM3\_FORCE\_RESET  
\_\_HAL\_RCC\_WWDG\_FORCE\_RESET  
\_\_HAL\_RCC\_USART2\_FORCE\_RESET  
\_\_HAL\_RCC\_I2C1\_FORCE\_RESET  
\_\_HAL\_RCC\_BKP\_FORCE\_RESET  
\_\_HAL\_RCC\_PWR\_FORCE\_RESET  
\_\_HAL\_RCC\_APB1\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM2\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM3\_RELEASE\_RESET  
\_\_HAL\_RCC\_WWDG\_RELEASE\_RESET  
\_\_HAL\_RCC\_USART2\_RELEASE\_RESET  
\_\_HAL\_RCC\_I2C1\_RELEASE\_RESET  
\_\_HAL\_RCC\_BKP\_RELEASE\_RESET  
\_\_HAL\_RCC\_PWR\_RELEASE\_RESET

**APB1 Peripheral Clock Enable Disable Status**

\_\_HAL\_RCC\_TIM2\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM2\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM3\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM3\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_WWDG\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_WWDG\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_USART2\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_USART2\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_I2C1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_I2C1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_BKP\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_BKP\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_PWR\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_PWR\_IS\_CLK\_DISABLED

***APB2 Clock Enable Disable***

\_\_HAL\_RCC\_AFIO\_CLK\_ENABLE  
\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE  
\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE  
\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE  
\_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE  
\_\_HAL\_RCC\_ADC1\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM1\_CLK\_ENABLE  
\_\_HAL\_RCC\_SPI1\_CLK\_ENABLE  
\_\_HAL\_RCC\_USART1\_CLK\_ENABLE  
\_\_HAL\_RCC\_AFIO\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOA\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOB\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOC\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOD\_CLK\_DISABLE  
\_\_HAL\_RCC\_ADC1\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM1\_CLK\_DISABLE  
\_\_HAL\_RCC\_SPI1\_CLK\_DISABLE  
\_\_HAL\_RCC\_USART1\_CLK\_DISABLE

***APB2 Force Release Reset***

\_\_HAL\_RCC\_APB2\_FORCE\_RESET  
\_\_HAL\_RCC\_AFIO\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOA\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOB\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOC\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOD\_FORCE\_RESET  
\_\_HAL\_RCC\_ADC1\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM1\_FORCE\_RESET  
\_\_HAL\_RCC\_SPI1\_FORCE\_RESET

\_\_HAL\_RCC\_USART1\_FORCE\_RESET  
\_\_HAL\_RCC\_APB2\_RELEASE\_RESET  
\_\_HAL\_RCC\_AFIO\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOA\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOB\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOC\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOD\_RELEASE\_RESET  
\_\_HAL\_RCC\_ADC1\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM1\_RELEASE\_RESET  
\_\_HAL\_RCC\_SPI1\_RELEASE\_RESET  
\_\_HAL\_RCC\_USART1\_RELEASE\_RESET

***APB2 Peripheral Clock Enable Disable Status***

\_\_HAL\_RCC\_AFIO\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_AFIO\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOA\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOA\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOB\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOB\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOC\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOC\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOD\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOD\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_ADC1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_ADC1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SPI1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SPI1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_USART1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_USART1\_IS\_CLK\_DISABLED

***BitAddress AliasRegion***

RCC\_OFFSET  
RCC\_CR\_OFFSET  
RCC\_CFGR\_OFFSET  
RCC\_CIR\_OFFSET  
RCC\_BDCR\_OFFSET

RCC\_CSR\_OFFSET  
RCC\_CR\_OFFSET\_BB  
RCC\_CFGR\_OFFSET\_BB  
RCC\_CIR\_OFFSET\_BB  
RCC\_BDCR\_OFFSET\_BB  
RCC\_CSR\_OFFSET\_BB  
HSION\_BITNUMBER  
RCC\_CR\_HSION\_BB  
HSEON\_BITNUMBER  
CR\_HSEON\_BB  
CSSON\_BITNUMBER  
RCC\_CR\_CSSON\_BB  
PLLON\_BITNUMBER  
RCC\_CR\_PLLON\_BB  
LSION\_BITNUMBER  
RCC\_CSR\_LSION\_BB  
LSEON\_BITNUMBER  
BDCR\_LSEON\_BB  
LSEBYP\_BITNUMBER  
BDCR\_LSEBYP\_BB  
RTCEN\_BITNUMBER  
RCC\_BDCR\_RTCEN\_BB  
BDRST\_BITNUMBER  
RCC\_BDCR\_BDRST\_BB  
RCC\_CR\_BYTE2\_ADDRESS  
RCC\_CIR\_BYTE1\_ADDRESS  
RCC\_CIR\_BYTE2\_ADDRESS  
CR\_REG\_INDEX  
BDCR\_REG\_INDEX  
CSR\_REG\_INDEX  
RCC\_FLAG\_MASK

**Flags**

RCC_FLAG_HSIRDY	Internal High Speed clock ready flag
RCC_FLAG_HSERDY	External High Speed clock ready flag
RCC_FLAG_PLLRDY	PLL clock ready flag
RCC_FLAG_LSERDY	External Low Speed oscillator Ready

RCC_FLAG_LSIRDY	Internal Low Speed oscillator Ready
RCC_FLAG_RMV	Remove reset flag
RCC_FLAG_PINRST	PIN reset flag
RCC_FLAG_PORRST	POR/PDR reset flag
RCC_FLAG_SFTRST	Software Reset flag
RCC_FLAG_IWDGRST	Independent Watchdog reset flag
RCC_FLAG_WWDGRST	Window watchdog reset flag
RCC_FLAG_LPWRRST	Low-Power reset flag

### Flags Interrupts Management

`__HAL_RCC_ENABLE_IT`

#### Description:

- Enable RCC interrupt (Perform Byte access to RCC\_CIR[14:8] bits to enable the selected interrupts.).

#### Parameters:

- `__INTERRUPT__`: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt
  - RCC\_IT\_LSERDY: LSE ready interrupt
  - RCC\_IT\_HSIRDY: HSI ready interrupt
  - RCC\_IT\_HSERDY: HSE ready interrupt
  - RCC\_IT\_PLLRDY: main PLL ready interrupt
  - RCC\_IT\_PLL2RDY: Main PLL2 ready interrupt.(\*)
  - RCC\_IT\_PLLI2S2RDY: Main PLLI2S ready interrupt.(\*)

`__HAL_RCC_DISABLE_IT`

#### Description:

- Disable RCC interrupt (Perform Byte access to RCC\_CIR[14:8] bits to disable the selected interrupts).

#### Parameters:

- `__INTERRUPT__`: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt
  - RCC\_IT\_LSERDY: LSE ready interrupt
  - RCC\_IT\_HSIRDY: HSI ready interrupt
  - RCC\_IT\_HSERDY: HSE ready interrupt
  - RCC\_IT\_PLLRDY: main PLL ready interrupt
  - RCC\_IT\_PLL2RDY: Main PLL2 ready

interrupt.(\*)

- RCC\_IT\_PLLI2S2RDY: Main PLLI2S ready interrupt.(\*)

#### \_\_HAL\_RCC\_CLEAR\_IT

##### Description:

- Clear the RCC's interrupt pending bits ( Perform Byte access to RCC\_CIR[23:16] bits to clear the selected interrupt pending bits.

##### Parameters:

- \_\_INTERRUPT\_\_: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt.
  - RCC\_IT\_LSERDY: LSE ready interrupt.
  - RCC\_IT\_HSIIRDY: HSI ready interrupt.
  - RCC\_IT\_HSERDY: HSE ready interrupt.
  - RCC\_IT\_PLLRDY: Main PLL ready interrupt.
  - RCC\_IT\_PLL2RDY: Main PLL2 ready interrupt.(\*)
  - RCC\_IT\_PLLI2S2RDY: Main PLLI2S ready interrupt.(\*)

#### \_\_HAL\_RCC\_GET\_IT

##### Description:

- Check the RCC's interrupt has occurred or not.

##### Parameters:

- \_\_INTERRUPT\_\_: specifies the RCC interrupt source to check. This parameter can be one of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt.
  - RCC\_IT\_LSERDY: LSE ready interrupt.
  - RCC\_IT\_HSIIRDY: HSI ready interrupt.
  - RCC\_IT\_HSERDY: HSE ready interrupt.
  - RCC\_IT\_PLLRDY: Main PLL ready interrupt.
  - RCC\_IT\_PLL2RDY: Main PLL2 ready interrupt.(\*)
  - RCC\_IT\_PLLI2S2RDY: Main PLLI2S ready interrupt.(\*)
  - RCC\_IT\_CSS: Clock Security System interrupt

##### Return value:

- The: new state of \_\_INTERRUPT\_\_ (TRUE or FALSE).

#### \_\_HAL\_RCC\_CLEAR\_RESET\_FLAGS

#### \_\_HAL\_RCC\_GET\_FLAG

##### Description:

- Check RCC flag is set or not.

**Parameters:**

- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `RCC_FLAG_HSIRDY`: HSI oscillator clock ready.
  - `RCC_FLAG_HSERDY`: HSE oscillator clock ready.
  - `RCC_FLAG_PLLRDY`: Main PLL clock ready.
  - `RCC_FLAG_PLL2RDY`: Main PLL2 clock ready.(\*)
  - `RCC_FLAG_PLLI2SRDY`: Main PLLI2S clock ready.(\*)
  - `RCC_FLAG_LSERDY`: LSE oscillator clock ready.
  - `RCC_FLAG_LSIRDY`: LSI oscillator clock ready.
  - `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.

**Return value:**

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

**Get Clock source**

`__HAL_RCC_GET_SYSCLK_SOURCE`  
RCE

**Description:**

- Macro to get the clock source used as system clock.

**Return value:**

- The: clock source used as system clock. The returned value can be one of the following:
  - `RCC_SYSCLKSOURCE_STATUS_HSI`: HSI used as system clock
  - `RCC_SYSCLKSOURCE_STATUS_HSE`: HSE used as system clock
  - `RCC_SYSCLKSOURCE_STATUS_PLLCLK`: PLL used as system clock

`__HAL_RCC_GET_PLL_OSCSOURCE`  
RCE

**Description:**

- Get oscillator clock selected as PLL input clock.



**Return value:**

- The: clock source used for PLL entry. The returned value can be one of the following:
  - RCC\_PLLSOURCE\_HSI\_DIV2: HSI oscillator clock selected as PLL input clock
  - RCC\_PLLSOURCE\_HSE: HSE oscillator clock selected as PLL input clock

**HSE Config**

RCC\_HSE\_OFF            HSE clock deactivation

RCC\_HSE\_ON            HSE clock activation

RCC\_HSE\_BYPASS    External clock source for HSE clock

**HSE Configuration**

\_\_HAL\_RCC\_HSE\_CONFIG    **Description:**

- Macro to configure the External High Speed oscillator (HSE).

**Parameters:**

- \_\_STATE\_\_: specifies the new state of the HSE. This parameter can be one of the following values:
  - RCC\_HSE\_OFF: turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
  - RCC\_HSE\_ON: turn ON the HSE oscillator
  - RCC\_HSE\_BYPASS: HSE oscillator bypassed with external clock

**HSI Config**

RCC\_HSI\_OFF            HSI clock deactivation

RCC\_HSI\_ON            HSI clock activation

RCC\_HSICALIBRATION\_DEFAULT

**HSI Configuration**

\_\_HAL\_RCC\_HSI\_ENABLE

\_\_HAL\_RCC\_HSI\_DISABLE

\_\_HAL\_RCC\_HSI\_CALIBRATIONVALUE\_ADJUST

**Description:**

- macro to adjust the Internal High Speed oscillator (HSI) calibration value.

**Parameters:**

- \_HSICALIBRATIONVALUE\_: specifies the calibration trimming value. (default is RCC\_HSICALIBRATION\_DEFAULT). This parameter must be a number between 0 and 0x1F.

**Interrupts**

---

RCC_IT_LSIRDY	LSI Ready Interrupt flag
RCC_IT_LSERDY	LSE Ready Interrupt flag
RCC_IT_HSIRDY	HSI Ready Interrupt flag
RCC_IT_HSERDY	HSE Ready Interrupt flag
RCC_IT_PLLRDY	PLL Ready Interrupt flag
RCC_IT_CSS	Clock Security System Interrupt flag

**LSE Config**

RCC_LSE_OFF	LSE clock deactivation
RCC_LSE_ON	LSE clock activation
RCC_LSE_BYPASS	External clock source for LSE clock

**LSE Configuration**

\_\_HAL\_RCC\_LSE\_CONFIG

**LSI Config**

RCC_LSI_OFF	LSI clock deactivation
RCC_LSI_ON	LSI clock activation

**LSI Configuration**

\_\_HAL\_RCC\_LSI\_ENABLE

\_\_HAL\_RCC\_LSI\_DISABLE

**MCO1 Clock Prescaler**

RCC\_MCODIV\_1

**MCO Index**

RCC\_MCO1

RCC\_MCO MCO1 to be compliant with other families with 2 MCOs

**Oscillator Type**

RCC\_OSCILLATORTYPE\_NONE

RCC\_OSCILLATORTYPE\_HSE

RCC\_OSCILLATORTYPE\_HSI

RCC\_OSCILLATORTYPE\_LSE

RCC\_OSCILLATORTYPE\_LSI

**Peripheral Clock Enable Disable**

\_\_HAL\_RCC\_DMA1\_CLK\_ENABLE

\_\_HAL\_RCC\_SRAM\_CLK\_ENABLE

\_\_HAL\_RCC\_FLITF\_CLK\_ENABLE

\_\_HAL\_RCC\_CRC\_CLK\_ENABLE

\_\_HAL\_RCC\_DMA1\_CLK\_DISABLE

\_\_HAL\_RCC\_SRAM\_CLK\_DISABLE

\_\_HAL\_RCC\_FLITF\_CLK\_DISABLE

\_\_HAL\_RCC\_CRC\_CLK\_DISABLE

### **PLL Clock Source**

RCC\_PLLSOURCE\_HSI\_DIV2 HSI clock divided by 2 selected as PLL entry clock source

RCC\_PLLSOURCE\_HSE HSE clock selected as PLL entry clock source

### **PLL Config**

RCC\_PLL\_NONE PLL is not configured

RCC\_PLL\_OFF PLL deactivation

RCC\_PLL\_ON PLL activation

### **PLL Configuration**

\_\_HAL\_RCC\_PLL\_ENABLE

\_\_HAL\_RCC\_PLL\_DISABLE

\_\_HAL\_RCC\_PLL\_CONFIG

#### **Description:**

- macros to configure the main PLL clock source and multiplication factors.

#### **Parameters:**

- \_\_RCC\_PLLSOURCE\_\_: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC\_PLLSOURCE\_HSI\_DIV2: HSI oscillator clock selected as PLL clock entry
  - RCC\_PLLSOURCE\_HSE: HSE oscillator clock selected as PLL clock entry
- \_\_PLLMUL\_\_: specifies the multiplication factor for PLL VCO output clock. This parameter can be one of the following values:
  - RCC\_PLL\_MUL2: PLLVCO = PLL clock entry x 2 (\*)
  - RCC\_PLL\_MUL3: PLLVCO = PLL clock entry x 3 (\*)
  - RCC\_PLL\_MUL4: PLLVCO = PLL clock entry x 4 (\*)
  - RCC\_PLL\_MUL6: PLLVCO = PLL clock entry x 6 (\*)
  - RCC\_PLL\_MUL6\_5: PLLVCO = PLL clock entry x 6.5 (\*\*)
  - RCC\_PLL\_MUL8: PLLVCO = PLL clock entry x 8 (\*)
  - RCC\_PLL\_MUL9: PLLVCO = PLL clock entry x 9 (\*)
  - RCC\_PLL\_MUL10: PLLVCO = PLL clock entry x 10 (\*)
  - RCC\_PLL\_MUL11: PLLVCO = PLL clock entry x 11 (\*)
  - RCC\_PLL\_MUL12: PLLVCO = PLL clock entry x 12 (\*)
  - RCC\_PLL\_MUL13: PLLVCO = PLL clock entry x 13 (\*)
  - RCC\_PLL\_MUL14: PLLVCO = PLL clock entry x 14 (\*)

- RCC\_PLL\_MUL15: PLLVCO = PLL clock entry x 15 (\*)
- RCC\_PLL\_MUL16: PLLVCO = PLL clock entry x 16 (\*)

**RCC Private Constants**

RCC\_DBP\_TIMEOUT\_VALUE  
 RCC\_LSE\_TIMEOUT\_VALUE  
 CLOCKSWITCH\_TIMEOUT\_VALUE  
 HSE\_TIMEOUT\_VALUE  
 HSI\_TIMEOUT\_VALUE  
 LSI\_TIMEOUT\_VALUE  
 PLL\_TIMEOUT\_VALUE  
 LSI\_VALUE

**RCC Private Macros**

MCO1\_CLK\_ENABLE  
 MCO1\_GPIO\_PORT  
 MCO1\_PIN  
 IS\_RCC\_HSI  
 IS\_RCC\_CALIBRATION\_VALUE  
 IS\_RCC\_CLOCKTYPE  
 IS\_RCC\_HSE  
 IS\_RCC\_LSE  
 IS\_RCC\_PLLSOURCE  
 IS\_RCC\_OSCILLATORTYPE  
 IS\_RCC\_LSI  
 IS\_RCC\_PLL  
 IS\_RCC\_SYSCLKSOURCE  
 IS\_RCC\_HCLK  
 IS\_RCC\_PCLK  
 IS\_RCC\_MCO  
 IS\_RCC\_MCODIV

**RCC RTC Clock Configuration**

\_\_HAL\_RCC\_RTC\_CONFIG

**Description:**

- Macro to configures the RTC clock (RTCCLK).

**Parameters:**

- \_\_RTC\_CLKSOURCE\_\_: specifies the RTC clock source. This parameter can be one of the following values:

- RCC\_RTCCLKSOURCE\_LSE: LSE selected as RTC clock
- RCC\_RTCCLKSOURCE\_LSI: LSI selected as RTC clock
- RCC\_RTCCLKSOURCE\_HSE\_DIV128 : HSE divided by 128 selected as RTC clock

\_\_HAL\_RCC\_GET\_RTC\_SOURCE

\_\_HAL\_RCC\_RTC\_ENABLE

\_\_HAL\_RCC\_RTC\_DISABLE

\_\_HAL\_RCC\_BACKUPRESET\_FORCE

\_\_HAL\_RCC\_BACKUPRESET\_RELEASE

#### **RTC Clock Source**

RCC\_RTCCLKSOURCE\_LSE            LSE oscillator clock used as RTC clock

RCC\_RTCCLKSOURCE\_LSI            LSI oscillator clock used as RTC clock

RCC\_RTCCLKSOURCE\_HSE\_DIV128    HSE oscillator clock divided by 128 used as RTC clock

#### **System Clock Source**

RCC\_SYSCLKSOURCE\_HSI            HSI selected as system clock

RCC\_SYSCLKSOURCE\_HSE            HSE selected as system clock

RCC\_SYSCLKSOURCE\_PLLCLK        PLL selected as system clock

#### **System Clock Source Status**

RCC\_SYSCLKSOURCE\_STATUS\_HSI

RCC\_SYSCLKSOURCE\_STATUS\_HSE

RCC\_SYSCLKSOURCE\_STATUS\_PLLCLK

#### **System Clock Type**

RCC\_CLOCKTYPE\_SYSCLK    SYSCLK to configure

RCC\_CLOCKTYPE\_HCLK        HCLK to configure

RCC\_CLOCKTYPE\_PCLK1       PCLK1 to configure

RCC\_CLOCKTYPE\_PCLK2       PCLK2 to configure

## 32 HAL RCC Extension Driver

### 32.1 RCCEX Firmware driver registers structures

#### 32.1.1 RCC\_OscInitTypeDef

*RCC\_OscInitTypeDef* is defined in the stm32f1xx\_hal\_rcc\_ex.h

##### Data Fields

- *uint32\_t OscillatorType*
- *uint32\_t HSEState*
- *uint32\_t HSEPredivValue*
- *uint32\_t LSEState*
- *uint32\_t HSIState*
- *uint32\_t HSCalibrationValue*
- *uint32\_t LSIState*
- *RCC\_PLLInitTypeDef PLL*

##### Field Documentation

- *uint32\_t RCC\_OscInitTypeDef::OscillatorType* The oscillators to be configured. This parameter can be a value of [RCC\\_Oscillator\\_Type](#)
- *uint32\_t RCC\_OscInitTypeDef::HSEState* The new state of the HSE. This parameter can be a value of [RCC\\_HSE\\_Config](#)
- *uint32\_t RCC\_OscInitTypeDef::HSEPredivValue* The Prediv1 factor value (named PREDIV1 or PLLXTPRE in RM) This parameter can be a value of [RCCEX\\_Prediv1\\_Factor](#)
- *uint32\_t RCC\_OscInitTypeDef::LSEState* The new state of the LSE. This parameter can be a value of [RCC\\_LSE\\_Config](#)
- *uint32\_t RCC\_OscInitTypeDef::HSIState* The new state of the HSI. This parameter can be a value of [RCC\\_HSI\\_Config](#)
- *uint32\_t RCC\_OscInitTypeDef::HSCalibrationValue* The HSI calibration trimming value (default is RCC\_HSCALIBRATION\_DEFAULT). This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F
- *uint32\_t RCC\_OscInitTypeDef::LSIState* The new state of the LSI. This parameter can be a value of [RCC\\_LSI\\_Config](#)
- *RCC\_PLLInitTypeDef RCC\_OscInitTypeDef::PLL* PLL structure parameters

#### 32.1.2 RCC\_PeriphCLKInitTypeDef

*RCC\_PeriphCLKInitTypeDef* is defined in the stm32f1xx\_hal\_rcc\_ex.h

##### Data Fields

- *uint32\_t PeriphClockSelection*
- *uint32\_t RTCClockSelection*
- *uint32\_t AdcClockSelection*
- *uint32\_t I2s2ClockSelection*
- *uint32\_t I2s3ClockSelection*
- *uint32\_t UsbClockSelection*

### Field Documentation

- **`uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection`** The Extended Clock to be configured. This parameter can be a value of [RCCEx\\_Periph\\_Clock\\_Selection](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection`** specifies the RTC clock source. This parameter can be a value of [RCC\\_RTC\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::AdcClockSelection`** ADC clock source This parameter can be a value of [RCCEx\\_ADC\\_Prescaler](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2s2ClockSelection`** I2S2 clock source This parameter can be a value of [RCCEx\\_I2S2\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2s3ClockSelection`** I2S3 clock source This parameter can be a value of [RCCEx\\_I2S3\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection`** USB clock source This parameter can be a value of [RCCEx\\_USB\\_Prescaler](#)

## 32.2 RCCEx Firmware driver API description

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

### 32.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.



Important note: Care must be taken when `HAL_RCCEx_PeriphCLKConfig()` is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and `RCC_BDCR` register are set to their reset values.

- [HAL\\_RCCEx\\_PeriphCLKConfig\(\)](#)
- [HAL\\_RCCEx\\_GetPeriphCLKConfig\(\)](#)
- [HAL\\_RCCEx\\_GetPeriphCLKFreq\(\)](#)

### 32.2.2 HAL\_RCCEx\_PeriphCLKConfig

Function Name	<b>HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)</b>
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"> <li>• <b>PeriphClkInit:</b> pointer to an <code>RCC_PeriphCLKInitTypeDef</code> structure that contains the configuration information for the Extended Peripherals clocks(RTC clock).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Care must be taken when <code>HAL_RCCEx_PeriphCLKConfig()</code> is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source,</li> </ul>

as consequence RTC registers (including the backup registers) are set to their reset values.

- In case of STM32F105xC or STM32F107xC devices, PLLI2S will be enabled if requested on one of 2 I2S interfaces. When PLLI2S is enabled, you need to call HAL\_RCCEx\_DisablePLLI2S to manually disable it.

### 32.2.3 HAL\_RCCEx\_GetPeriphCLKConfig

Function Name	<b>void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)</b>
Function Description	Get the PeriphClkInit according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClkInit:</b> pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks(RTC, I2S, ADC clocks).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 32.2.4 HAL\_RCCEx\_GetPeriphCLKFreq

Function Name	<b>uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint32_t PeriphClk)</b>
Function Description	Returns the peripheral clock frequency.
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClk:</b> Peripheral clock identifier This parameter can be one of the following values: RCC_PERIPHCLK_RTC: RTC peripheral clock RCC_PERIPHCLK_ADC: ADC peripheral clock RCC_PERIPHCLK_I2S2: I2S2 peripheral clock (STM32F103xE, STM32F103xG, STM32F105xC &amp; STM32F107xC) RCC_PERIPHCLK_I2S3: I2S3 peripheral clock (STM32F103xE, STM32F103xG, STM32F105xC &amp; STM32F107xC) RCC_PERIPHCLK_USB: USB peripheral clock (STM32F102xx, STM32F103xx, STM32F105xC &amp; STM32F107xC)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Frequency in Hz (0: means that no available frequency for the peripheral)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Returns 0 if peripheral clock is unknown</li> </ul>

## 32.3 RCCEx Firmware driver defines

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

### 32.3.1 RCCEx

RCCEx

**ADC Prescaler**

RCC\_ADCPCLK2\_DIV2

RCC\_ADCPCLK2\_DIV4



RCC\_ADCPCLK2\_DIV6

RCC\_ADCPCLK2\_DIV8

***AHB1 Peripheral Clock Enable Disable Status***

\_\_HAL\_RCC\_DMA2\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_DMA2\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_FSMC\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_FSMC\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_SDIO\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_SDIO\_IS\_CLK\_DISABLED

***APB1 Clock Enable Disable***

\_\_HAL\_RCC\_CAN1\_CLK\_ENABLE

\_\_HAL\_RCC\_CAN1\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM4\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI2\_CLK\_ENABLE

\_\_HAL\_RCC\_USART3\_CLK\_ENABLE

\_\_HAL\_RCC\_I2C2\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM4\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI2\_CLK\_DISABLE

\_\_HAL\_RCC\_USART3\_CLK\_DISABLE

\_\_HAL\_RCC\_I2C2\_CLK\_DISABLE

\_\_HAL\_RCC\_USB\_CLK\_ENABLE

\_\_HAL\_RCC\_USB\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM5\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM6\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM7\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI3\_CLK\_ENABLE

\_\_HAL\_RCC\_UART4\_CLK\_ENABLE

\_\_HAL\_RCC\_UART5\_CLK\_ENABLE

\_\_HAL\_RCC\_DAC\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM5\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM6\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM7\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI3\_CLK\_DISABLE

\_\_HAL\_RCC\_UART4\_CLK\_DISABLE

\_\_HAL\_RCC\_UART5\_CLK\_DISABLE

\_\_HAL\_RCC\_DAC\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM12\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM13\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM14\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM12\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM13\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM14\_CLK\_DISABLE

**APB1 Force Release Reset**

\_\_HAL\_RCC\_CAN1\_FORCE\_RESET  
\_\_HAL\_RCC\_CAN1\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM4\_FORCE\_RESET  
\_\_HAL\_RCC\_SPI2\_FORCE\_RESET  
\_\_HAL\_RCC\_USART3\_FORCE\_RESET  
\_\_HAL\_RCC\_I2C2\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM4\_RELEASE\_RESET  
\_\_HAL\_RCC\_SPI2\_RELEASE\_RESET  
\_\_HAL\_RCC\_USART3\_RELEASE\_RESET  
\_\_HAL\_RCC\_I2C2\_RELEASE\_RESET  
\_\_HAL\_RCC\_USB\_FORCE\_RESET  
\_\_HAL\_RCC\_USB\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM5\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM6\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM7\_FORCE\_RESET  
\_\_HAL\_RCC\_SPI3\_FORCE\_RESET  
\_\_HAL\_RCC\_UART4\_FORCE\_RESET  
\_\_HAL\_RCC\_UART5\_FORCE\_RESET  
\_\_HAL\_RCC\_DAC\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM5\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM6\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM7\_RELEASE\_RESET  
\_\_HAL\_RCC\_SPI3\_RELEASE\_RESET  
\_\_HAL\_RCC\_UART4\_RELEASE\_RESET  
\_\_HAL\_RCC\_UART5\_RELEASE\_RESET  
\_\_HAL\_RCC\_DAC\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM12\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM13\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM14\_FORCE\_RESET

\_\_HAL\_RCC\_TIM12\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM13\_RELEASE\_RESET

\_\_HAL\_RCC\_TIM14\_RELEASE\_RESET

***APB1 Peripheral Clock Enable Disable Status***

\_\_HAL\_RCC\_CAN1\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_CAN1\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_TIM4\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_TIM4\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_SPI2\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_SPI2\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_USART3\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_USART3\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_I2C2\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_I2C2\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_USB\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_USB\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_TIM5\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_TIM5\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_TIM6\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_TIM6\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_TIM7\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_TIM7\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_SPI3\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_SPI3\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_UART4\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_UART4\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_UART5\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_UART5\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_DAC\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_DAC\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_TIM13\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_TIM13\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_TIM14\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_TIM14\_IS\_CLK\_DISABLED

***APB2 Clock Enable Disable***

\_\_HAL\_RCC\_ADC2\_CLK\_ENABLE

\_\_HAL\_RCC\_ADC2\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOE\_CLK\_ENABLE  
\_\_HAL\_RCC\_GPIOE\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOF\_CLK\_ENABLE  
\_\_HAL\_RCC\_GPIOG\_CLK\_ENABLE  
\_\_HAL\_RCC\_GPIOF\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOG\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM8\_CLK\_ENABLE  
\_\_HAL\_RCC\_ADC3\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM8\_CLK\_DISABLE  
\_\_HAL\_RCC\_ADC3\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM9\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM10\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM11\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM9\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM10\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM11\_CLK\_DISABLE

**APB2 Force Release Reset**

\_\_HAL\_RCC\_ADC2\_FORCE\_RESET  
\_\_HAL\_RCC\_ADC2\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOE\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOE\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOF\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOG\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOF\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOG\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM8\_FORCE\_RESET  
\_\_HAL\_RCC\_ADC3\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM8\_RELEASE\_RESET  
\_\_HAL\_RCC\_ADC3\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM9\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM10\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM11\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM9\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM10\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM11\_RELEASE\_RESET

**APB2 Peripheral Clock Enable Disable Status**

```

__HAL_RCC_ADC2_IS_CLK_ENABLED
__HAL_RCC_ADC2_IS_CLK_DISABLED
__HAL_RCC_GPIOE_IS_CLK_ENABLED
__HAL_RCC_GPIOE_IS_CLK_DISABLED
__HAL_RCC_GPIOF_IS_CLK_ENABLED
__HAL_RCC_GPIOF_IS_CLK_DISABLED
__HAL_RCC_GPIOG_IS_CLK_ENABLED
__HAL_RCC_GPIOG_IS_CLK_DISABLED
__HAL_RCC_TIM8_IS_CLK_ENABLED
__HAL_RCC_TIM8_IS_CLK_DISABLED
__HAL_RCC_ADC3_IS_CLK_ENABLED
__HAL_RCC_ADC3_IS_CLK_DISABLED
__HAL_RCC_TIM9_IS_CLK_ENABLED
__HAL_RCC_TIM9_IS_CLK_DISABLED
__HAL_RCC_TIM10_IS_CLK_ENABLED
__HAL_RCC_TIM10_IS_CLK_DISABLED
__HAL_RCC_TIM11_IS_CLK_ENABLED
__HAL_RCC_TIM11_IS_CLK_DISABLED

```

#### **HSE Configuration**

`__HAL_RCC_HSE_PREDIV_CONFIG` **Description:**

- Macro to configure the External High Speed oscillator (HSE) Predivision factor for PLL.

#### **Parameters:**

- `__HSE_PREDIV_VALUE__`: specifies the division value applied to HSE. This parameter must be a number between `RCC_HSE_PREDIV_DIV1` and `RCC_HSE_PREDIV_DIV2`.

```
__HAL_RCC_HSE_GET_PREDIV
```

#### **I2S2 Clock Source**

```
RCC_I2S2CLKSOURCE_SYSCLK
```

#### **I2S3 Clock Source**

```
RCC_I2S3CLKSOURCE_SYSCLK
```

#### **MCO1 Clock Source**

```
RCC_MCO1SOURCE_NOCLOCK
```

```
RCC_MCO1SOURCE_SYSCLK
```

```
RCC_MCO1SOURCE_HSI
```

```
RCC_MCO1SOURCE_HSE
```

RCC\_MCO1SOURCE\_PLLCLK

**Peripheral Clock Enable Disable**

\_\_HAL\_RCC\_DMA2\_CLK\_ENABLE

\_\_HAL\_RCC\_DMA2\_CLK\_DISABLE

\_\_HAL\_RCC\_FSMC\_CLK\_ENABLE

\_\_HAL\_RCC\_FSMC\_CLK\_DISABLE

\_\_HAL\_RCC\_SDIO\_CLK\_ENABLE

\_\_HAL\_RCC\_SDIO\_CLK\_DISABLE

**Peripheral Configuration**

\_\_HAL\_RCC\_USB\_CONFIG

**Description:**

- Macro to configure the USB clock.

**Parameters:**

- \_\_USBCLKSOURCE\_\_: specifies the USB clock source. This parameter can be one of the following values:
  - RCC\_USBPLLCLK\_DIV1: PLL clock divided by 1 selected as USB clock
  - RCC\_USBPLLCLK\_DIV1\_5: PLL clock divided by 1.5 selected as USB clock

\_\_HAL\_RCC\_GET\_USB\_SOURCE

**Description:**

- Macro to get the USB clock (USBCLK).

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_USBPLLCLK\_DIV1: PLL clock divided by 1 selected as USB clock
  - RCC\_USBPLLCLK\_DIV1\_5: PLL clock divided by 1.5 selected as USB clock

\_\_HAL\_RCC\_ADC\_CONFIG

**Description:**

- Macro to configure the ADCx clock (x=1 to 3 depending on devices).

**Parameters:**

- \_\_ADCCLKSOURCE\_\_: specifies the ADC clock source. This parameter can be one of the following values:
  - RCC\_ADCCLK2\_DIV2: PCLK2 clock divided by 2 selected as ADC clock
  - RCC\_ADCCLK2\_DIV4: PCLK2 clock divided by 4 selected as ADC clock
  - RCC\_ADCCLK2\_DIV6: PCLK2 clock divided by 6 selected as ADC clock
  - RCC\_ADCCLK2\_DIV8: PCLK2 clock divided by 8 selected as ADC clock

**\_\_HAL\_RCC\_GET\_ADC\_SOURCE****Description:**

- Macro to get the ADC clock (ADCxCLK, x=1 to 3 depending on devices).

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_ADCPCLK2\_DIV2: PCLK2 clock divided by 2 selected as ADC clock
  - RCC\_ADCPCLK2\_DIV4: PCLK2 clock divided by 4 selected as ADC clock
  - RCC\_ADCPCLK2\_DIV6: PCLK2 clock divided by 6 selected as ADC clock
  - RCC\_ADCPCLK2\_DIV8: PCLK2 clock divided by 8 selected as ADC clock

***Periph Clock Selection***

RCC\_PERIPHCLK\_RTC

RCC\_PERIPHCLK\_ADC

RCC\_PERIPHCLK\_I2S2

RCC\_PERIPHCLK\_I2S3

RCC\_PERIPHCLK\_USB

***PLL Multiplication Factor***

RCC\_PLL\_MUL2

RCC\_PLL\_MUL3

RCC\_PLL\_MUL4

RCC\_PLL\_MUL5

RCC\_PLL\_MUL6

RCC\_PLL\_MUL7

RCC\_PLL\_MUL8

RCC\_PLL\_MUL9

RCC\_PLL\_MUL10

RCC\_PLL\_MUL11

RCC\_PLL\_MUL12

RCC\_PLL\_MUL13

RCC\_PLL\_MUL14

RCC\_PLL\_MUL15

RCC\_PLL\_MUL16

***HSE Prediv1 Factor***

RCC\_HSE\_PREDIV\_DIV1

RCC\_HSE\_PREDIV\_DIV2

***RCCEX Private Constants***

PLL2\_TIMEOUT\_VALUE

PLL2ON\_BITNUMBER

CR\_PLL2ON\_BB

CR\_REG\_INDEX

***RCCEX Private Macros***

IS\_RCC\_HSE\_PREDIV

IS\_RCC\_PLL\_MUL

IS\_RCC\_MCO1SOURCE

IS\_RCC\_ADCPLLCLK\_DIV

IS\_RCC\_I2S2CLKSOURCE

IS\_RCC\_I2S3CLKSOURCE

IS\_RCC\_PERIPHCLK

IS\_RCC\_USBPLLCLK\_DIV

***USB Prescaler***

RCC\_USBPLLCLK\_DIV1

RCC\_USBPLLCLK\_DIV1\_5



## 33 HAL RTC Generic Driver

### 33.1 RTC Firmware driver registers structures

#### 33.1.1 RTC\_TimeTypeDef

*RTC\_TimeTypeDef* is defined in the stm32f1xx\_hal\_rtc.h

##### Data Fields

- *uint8\_t Hours*
- *uint8\_t Minutes*
- *uint8\_t Seconds*

##### 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 = 23
- *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

#### 33.1.2 RTC\_AlarmTypeDef

*RTC\_AlarmTypeDef* is defined in the stm32f1xx\_hal\_rtc.h

##### Data Fields

- *RTC\_TimeTypeDef AlarmTime*
- *uint32\_t Alarm*

##### Field Documentation

- *RTC\_TimeTypeDef RTC\_AlarmTypeDef::AlarmTime* Specifies the RTC Alarm Time members
- *uint32\_t RTC\_AlarmTypeDef::Alarm* Specifies the alarm ID (only 1 alarm ID for STM32F1). This parameter can be a value of [RTC\\_Alarms\\_Definitions](#)

#### 33.1.3 RTC\_InitTypeDef

*RTC\_InitTypeDef* is defined in the stm32f1xx\_hal\_rtc.h

##### Data Fields

- *uint32\_t AsynchPrediv*
- *uint32\_t OutPut*

##### Field Documentation

- ***uint32\_t RTC\_InitTypeDef::AsynchPrediv*** Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFFFFF or RTC\_AUTO\_1\_SECOND If RTC\_AUTO\_1\_SECOND is selected, AsynchPrediv will be set automatically to get 1sec timebase
- ***uint32\_t RTC\_InitTypeDef::Output*** Specifies which signal will be routed to the RTC Tamper pin. This parameter can be a value of [RTC\\_output\\_source\\_to\\_output\\_on\\_the\\_Tamper\\_pin](#)

### 33.1.4 RTC\_DateTypeDef

***RTC\_DateTypeDef*** is defined in the stm32f1xx\_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 (not necessary for HAL\_RTC\_SetDate). 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

### 33.1.5 RTC\_HandleTypeDef

***RTC\_HandleTypeDef*** is defined in the stm32f1xx\_hal\_rtc.h

#### Data Fields

- ***RTC\_TypeDef \* Instance***
- ***RTC\_InitTypeDef Init***
- ***RTC\_DateTypeDef DateToUpdate***
- ***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
- ***RTC\_DateTypeDef RTC\_HandleTypeDef::DateToUpdate*** Current date set by user and updated automatically
- ***HAL\_LockTypeDef RTC\_HandleTypeDef::Lock*** RTC locking object
- ***\_\_IO HAL\_RTCStateTypeDef RTC\_HandleTypeDef::State*** Time communication state

## 33.2 RTC Firmware driver API description

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

### 33.2.1 How to use this driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous prescaler to generate RTC 1Hz time base) 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.

#### Tamper configuration

- Enable the RTC Tamper and configure the Tamper Level using the HAL\_RTCEx\_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL\_RTCEx\_SetTamper\_IT() function.
- The TAMPER1 alternate function can be mapped to PC13

#### Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL\_RTCEx\_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL\_RTCEx\_BKUPRead() function.

### 33.2.2 WARNING: Drivers Restrictions

RTC version used on STM32F1 families is version V1. All the features supported by V2 (other families) will be not supported on F1.

As on V2, main RTC features are managed by HW. But on F1, date feature is completely managed by SW.

Then, there are some restrictions compared to other families:

- Only format 24 hours supported in HAL (format 12 hours not supported)
- Date is saved in SRAM. Then, when MCU is in STOP or STANDBY mode, date will be lost. User should implement a way to save date before entering in low power mode (an example is provided with firmware package based on backup registers)
- Date is automatically updated each time a HAL\_RTC\_GetTime or HAL\_RTC\_GetDate is called.
- Alarm detection is limited to 1 day. It will expire only 1 time (no alarm repetition, need to program a new alarm)

### 33.2.3 Backup Domain Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

To allow the RTC operating even when the main digital supply (VDD) is turned off, the VBAT pin powers the following blocks:

- The RTC
- The LSE oscillator
- PC13 I/O

When the backup domain is supplied by VDD (analog switch connected to VDD), the following pins are available:

- PC13 can be used as a Tamper pin

When the backup domain is supplied by VBAT (analog switch connected to VBAT because VDD is not present), the following pins are available:

- PC13 can be used as the Tamper pin

### 33.2.4 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC\_BDCR register to their reset values.

A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC\_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.
3. Tamper detection event resets all data backup registers.

### 33.2.5 Backup Domain Access

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

- Call the function HAL\_RCCEx\_PeriphCLKConfig in using RCC\_PERIPHCLK\_RTC for PeriphClockSelection and select RTCClockSelection (LSE, LSI or HSE)
- Enable the BKP clock in using \_\_HAL\_RCC\_BKP\_CLK\_ENABLE()

### 33.2.6 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 RTC tamper 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.

### 33.2.7 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Asynchronous), 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 should be programmed to generate the RTC 1Hz time base.
  2. All RTC registers are Write protected. Writing to the RTC registers is enabled by setting the CNF bit in the RTC\_CRL register.
  3. To read the calendar after wakeup from low power modes (Standby or Stop) the software must first wait for the RSF bit (Register Synchronized Flag) in the RTC\_CRL register to be set by hardware. The HAL\_RTC\_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).
- [HAL\\_RTC\\_Init\(\)](#)
  - [HAL\\_RTC\\_DeInit\(\)](#)
  - [HAL\\_RTC\\_Msplnit\(\)](#)
  - [HAL\\_RTC\\_MspDeInit\(\)](#)

### 33.2.8 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

- [HAL\\_RTC\\_SetTime\(\)](#)
- [HAL\\_RTC\\_GetTime\(\)](#)
- [HAL\\_RTC\\_SetDate\(\)](#)
- [HAL\\_RTC\\_GetDate\(\)](#)

### 33.2.9 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

- [HAL\\_RTC\\_SetAlarm\(\)](#)
- [HAL\\_RTC\\_SetAlarm\\_IT\(\)](#)
- [HAL\\_RTC\\_GetAlarm\(\)](#)
- [HAL\\_RTC\\_DeactivateAlarm\(\)](#)
- [HAL\\_RTC\\_AlarmIRQHandler\(\)](#)
- [HAL\\_RTC\\_AlarmAEventCallback\(\)](#)
- [HAL\\_RTC\\_PollForAlarmAEvent\(\)](#)

### 33.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state
- [HAL\\_RTC\\_GetState\(\)](#)

### 33.2.11 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization
- [HAL\\_RTC\\_WaitForSynchro\(\)](#)

### 33.2.12 HAL\_RTC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)</b>
Function Description	Initializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 33.2.13 HAL\_RTC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)</b>
Function Description	DeInitializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>
Notes	<ul style="list-style-type: none"><li>• This function does not reset the RTC Backup Data registers.</li></ul>

### 33.2.14 HAL\_RTC\_MspInit

Function Name	<b>void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)</b>
Function Description	Initializes the RTC MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 33.2.15 HAL\_RTC\_MspDeInit

Function Name	<b>void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)</b>
Function Description	DeInitializes the RTC MSP.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 33.2.16 HAL\_RTC\_SetTime

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTCimeTypeDef * sTime, uint32_t Format)</b>
Function Description	Sets RTC current time.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>sTime</b>: Pointer to Time structure</li> <li><b>Format</b>: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data format RTC_FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 33.2.17 HAL\_RTC\_GetTime

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTCimeTypeDef * sTime, uint32_t Format)</b>
Function Description	Gets RTC current time.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>sTime</b>: Pointer to Time structure</li> <li><b>Format</b>: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data format RTC_FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 33.2.18 HAL\_RTC\_SetDate

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)</b>
Function Description	Sets RTC current date.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>sDate</b>: Pointer to date structure</li> <li><b>Format</b>: specifies the format of the entered parameters. This parameter can be one of the following values:</li> </ul>

RTC\_FORMAT\_BIN: Binary data format  
RTC\_FORMAT\_BCD: BCD data format

Return values

- HAL status

### 33.2.19 HAL\_RTC\_GetDate

**Function Name** **HAL\_StatusTypeDef HAL\_RTC\_GetDate (RTC\_HandleTypeDef \* hrtc, RTC\_DateTypeDef \* sDate, uint32\_t Format)**

**Function Description** Gets RTC current date.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **sDate**: Pointer to Date structure
- **Format**: Specifies the format of the entered parameters. This parameter can be one of the following values:  
RTC\_FORMAT\_BIN: Binary data format  
RTC\_FORMAT\_BCD: BCD data format

Return values

- HAL status

### 33.2.20 HAL\_RTC\_SetAlarm

**Function Name** **HAL\_StatusTypeDef HAL\_RTC\_SetAlarm (RTC\_HandleTypeDef \* hrtc, RTC\_AlarmTypeDef \* sAlarm, uint32\_t Format)**

**Function Description** Sets the specified RTC Alarm.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **sAlarm**: Pointer to Alarm structure
- **Format**: Specifies the format of the entered parameters. This parameter can be one of the following values:  
RTC\_FORMAT\_BIN: Binary data format  
RTC\_FORMAT\_BCD: BCD data format

Return values

- HAL status

### 33.2.21 HAL\_RTC\_SetAlarm\_IT

**Function Name** **HAL\_StatusTypeDef HAL\_RTC\_SetAlarm\_IT (RTC\_HandleTypeDef \* hrtc, RTC\_AlarmTypeDef \* sAlarm, uint32\_t Format)**

**Function Description** Sets the specified RTC Alarm with Interrupt.

**Parameters**

- **hrtc**: pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **sAlarm**: Pointer to Alarm structure
- **Format**: Specifies the format of the entered parameters. This parameter can be one of the following values:  
RTC\_FORMAT\_BIN: Binary data format



RTC\_FORMAT\_BCD: BCD data format

- |               |   |
|---------------|---|
| Return values | • HAL status  |
| Notes         | • The HAL_RTC_SetTime() must be called before enabling the Alarm feature. |

### 33.2.22 HAL\_RTC\_GetAlarm

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_RTC_GetAlarm</b><br>(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"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sAlarm</b>: Pointer to Date structure</li> <li>• <b>Alarm</b>: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: Alarm</li> <li>• <b>Format</b>: Specifies the format of the entered parameters. This parameter can be one of the following values:<br/>RTC_FORMAT_BIN: Binary data format<br/>RTC_FORMAT_BCD: BCD data format</li> </ul> |
| Return values        | • HAL status   |

### 33.2.23 HAL\_RTC\_DeactivateAlarm

- |                      |   |
|----------------------|---|
| Function Name        | <b>HAL_StatusTypeDef HAL_RTC_DeactivateAlarm</b><br>(RTC_HandleTypeDef * hrtc, uint32_t Alarm)  |
| Function Description | Deactive the specified RTC Alarm.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Alarm</b>: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmA</li> </ul> |
| Return values        | • HAL status  |

### 33.2.24 HAL\_RTC\_AlarmIRQHandler

- |                      |  |
|----------------------|--|
| Function Name        | <b>void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)</b>   |
| Function Description | This function handles Alarm interrupt request.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul> |
| Return values        | • None   |

### 33.2.25 HAL\_RTC\_AlarmAEventCallback

Function Name	<b>void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Alarm A callback.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 33.2.26 HAL\_RTC\_PollForAlarmAEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	This function handles AlarmA Polling request.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 33.2.27 HAL\_RTC\_GetState

Function Name	<b>HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)</b>
Function Description	Returns the RTC state.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

### 33.2.28 HAL\_RTC\_WaitForSynchro

Function Name	<b>HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)</b>
Function Description	Waits until the RTC registers (RTC_CNT, RTC_ALR and RTC_PRL) are synchronized with RTC APB clock.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function must be called before any read operation after an APB reset or an APB clock stop.</li> </ul>

## 33.3 RTC Firmware driver defines

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

**33.3.1 RTC**

RTC

**Alarms Definitions**

RTC\_ALARM\_A Specify alarm ID (mainly for legacy purposes)

**Automatic calculation of prediv for 1sec timebase**

RTC\_AUTO\_1\_SECOND

**RTC Exported Macros**`__HAL_RTC_RESET_HANDLE_STATE`**Description:**

- Reset RTC handle state.

**Parameters:**

- `__HANDLE__`: RTC handle.

**Return value:**

- None:

`__HAL_RTC_WRITEPROTECTION_DISABLE`**Description:**

- Disable the write protection for RTC registers.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

`__HAL_RTC_WRITEPROTECTION_ENABLE`**Description:**

- Enable the write protection for RTC registers.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

`__HAL_RTC_ALARM_ENABLE_IT`**Description:**

- Enable the RTC Alarm interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Alarm

interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:

- RTC\_IT\_ALRA: Alarm A interrupt

**Return value:**

- None:

**Description:**

- Disable the RTC Alarm interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC\_IT\_ALRA: Alarm A interrupt

**Return value:**

- None:

**Description:**

- Check whether the specified RTC Alarm interrupt has been enabled or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC Alarm interrupt sources to be checked. This parameter can be:
  - RTC\_IT\_ALRA: Alarm A interrupt

**Return value:**

- None:

**Description:**

- Get the selected RTC Alarm's flag status.

\_\_HAL\_RTC\_ALARM\_DISABLE\_IT

\_\_HAL\_RTC\_ALARM\_GET\_IT\_SOURCE

\_\_HAL\_RTC\_ALARM\_GET\_FLAG

`__HAL_RTC_ALARM_GET_IT`

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_ALRAF`

**Return value:**

- None:

**Description:**

- Check whether the specified RTC Alarm interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - `RTC_IT_ALRA`: Alarm A interrupt

**Return value:**

- None:

**Description:**

- Clear the RTC Alarm's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_ALRAF`

**Return value:**

- None:

**Description:**

- Enable interrupt on ALARM Exti Line 17.

`__HAL_RTC_ALARM_CLEAR_FLAG`

`__HAL_RTC_ALARM_EXTI_ENABLE_IT`

\_\_HAL\_RTC\_ALARM\_EXTI\_DISABLE\_IT

**Return value:**

- None.:

**Description:**

- Disable interrupt on ALARM Exti Line 17.

**Return value:**

- None.:

**Description:**

- Enable event on ALARM Exti Line 17.

**Return value:**

- None.:

**Description:**

- Disable event on ALARM Exti Line 17.

**Return value:**

- None.:

**Description:**

- ALARM EXTI line configuration: set falling edge trigger.

**Return value:**

- None.:

**Description:**

- Disable the ALARM Extended Interrupt Falling Trigger.

**Return value:**

- None.:

**Description:**

- ALARM EXTI line configuration: set rising edge trigger.

**Return value:**

- None.:

**Description:**

- Disable the ALARM Extended Interrupt Rising Trigger.

**Return value:**

\_\_HAL\_RTC\_ALARM\_EXTI\_ENABLE\_EVENT

\_\_HAL\_RTC\_ALARM\_EXTI\_DISABLE\_EVENT

\_\_HAL\_RTC\_ALARM\_EXTI\_ENABLE\_FALLING\_EDGE

\_\_HAL\_RTC\_ALARM\_EXTI\_DISABLE\_FALLING\_EDGE

\_\_HAL\_RTC\_ALARM\_EXTI\_ENABLE\_RISING\_EDGE

\_\_HAL\_RTC\_ALARM\_EXTI\_DISABLE\_RISING\_EDGE

`__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE`

- None.:

**Description:**

- ALARM EXTI line configuration: set rising & falling edge trigger.

**Return value:**

- None.:

`__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE`

**Description:**

- Disable the ALARM Extended Interrupt Rising & Falling Trigger.

**Return value:**

- None.:

`__HAL_RTC_ALARM_EXTI_GET_FLAG`

**Description:**

- Check whether the specified ALARM EXTI interrupt flag is set or not.

**Return value:**

- EXTI: ALARM Line Status.

`__HAL_RTC_ALARM_EXTI_CLEAR_FLAG`

**Description:**

- Clear the ALARM EXTI flag.

**Return value:**

- None.:

`__HAL_RTC_ALARM_EXTI_GENERATE_SWIT`

**Description:**

- Generate a Software interrupt on selected EXTI line.

**Return value:**

- None.:

**RTC EXTI Line event**

`RTC_EXTI_LINE_ALARM_EVENT` External interrupt line 17 Connected to the RTC Alarm event

**Flags Definitions**

<code>RTC_FLAG_RTOFF</code>	RTC Operation OFF flag
<code>RTC_FLAG_RSIF</code>	Registers Synchronized flag
<code>RTC_FLAG_OW</code>	Overflow flag
<code>RTC_FLAG_ALRAF</code>	Alarm flag
<code>RTC_FLAG_SEC</code>	Second flag

RTC\_FLAG\_TAMP1F Tamper Interrupt Flag

**Input Parameter Format**

RTC\_FORMAT\_BIN

RTC\_FORMAT\_BCD

**Interrupts Definitions**

RTC\_IT\_OW Overflow interrupt

RTC\_IT\_ALRA Alarm interrupt

RTC\_IT\_SEC Second interrupt

RTC\_IT\_TAMP1 TAMPER Pin interrupt enable

**Month Definitions**

RTC\_MONTH\_JANUARY

RTC\_MONTH\_FEBRUARY

RTC\_MONTH\_MARCH

RTC\_MONTH\_APRIL

RTC\_MONTH\_MAY

RTC\_MONTH\_JUNE

RTC\_MONTH\_JULY

RTC\_MONTH\_AUGUST

RTC\_MONTH\_SEPTMBER

RTC\_MONTH\_OCTOBER

RTC\_MONTH\_NOVEMBER

RTC\_MONTH\_DECEMBER

**Output source to output on the Tamper pin**

RTC\_OUTPUTSOURCE\_NONE No output on the TAMPER pin

RTC\_OUTPUTSOURCE\_CALIBCLOCK RTC clock with a frequency divided by 64 on the TAMPER pin

RTC\_OUTPUTSOURCE\_ALARM Alarm pulse signal on the TAMPER pin

RTC\_OUTPUTSOURCE\_SECOND Second pulse signal on the TAMPER pin

**RTC Private Constants**

RTC\_ALARM\_RESETVALUE\_REGISTER

RTC\_ALARM\_RESETVALUE

**RTC Private Macros**

IS\_RTC\_ASYNCH\_PREDIV

IS\_RTC\_HOUR24

IS\_RTC\_MINUTES

IS\_RTC\_SECONDS

IS\_RTC\_FORMAT



IS\_RTC\_YEAR

IS\_RTC\_MONTH

IS\_RTC\_DATE

IS\_RTC\_ALARM

IS\_RTC\_CALIB\_OUTPUT

***Default Timeout Value***

RTC\_TIMEOUT\_VALUE

***WeekDay Definitions***

RTC\_WEEKDAY\_MONDAY

RTC\_WEEKDAY\_TUESDAY

RTC\_WEEKDAY\_WEDNESDAY

RTC\_WEEKDAY\_THURSDAY

RTC\_WEEKDAY\_FRIDAY

RTC\_WEEKDAY\_SATURDAY

RTC\_WEEKDAY\_SUNDAY

## 34 HAL RTC Extension Driver

### 34.1 RTCEX Firmware driver registers structures

#### 34.1.1 RTC\_TamperTypeDef

*RTC\_TamperTypeDef* is defined in the `stm32f1xx_hal_rtc_ex.h`

##### Data Fields

- *uint32\_t* *Tamper*
- *uint32\_t* *Trigger*

##### Field Documentation

- *uint32\_t* *RTC\_TamperTypeDef::Tamper* Specifies the Tamper Pin. This parameter can be a value of [RTCEX\\_Tamper\\_Pins\\_Definitions](#)
- *uint32\_t* *RTC\_TamperTypeDef::Trigger* Specifies the Tamper Trigger. This parameter can be a value of [RTCEX\\_Tamper\\_Trigger\\_Definitions](#)

### 34.2 RTCEX Firmware driver API description

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

#### 34.2.1 RTC Tamper functions

This section provides functions allowing to configure Tamper feature

- [HAL\\_RTCEX\\_SetTamper\(\)](#)
- [HAL\\_RTCEX\\_SetTamper\\_IT\(\)](#)
- [HAL\\_RTCEX\\_DeactivateTamper\(\)](#)
- [HAL\\_RTCEX\\_TamperIRQHandler\(\)](#)
- [HAL\\_RTCEX\\_Tamper1EventCallback\(\)](#)
- [HAL\\_RTCEX\\_PollForTamper1Event\(\)](#)

#### 34.2.2 RTC Second functions

This section provides functions implementing second interrupt handlers

- [HAL\\_RTCEX\\_SetSecond\\_IT\(\)](#)
- [HAL\\_RTCEX\\_DeactivateSecond\(\)](#)
- [HAL\\_RTCEX\\_RTCIRQHandler\(\)](#)
- [HAL\\_RTCEX\\_RTCEventCallback\(\)](#)
- [HAL\\_RTCEX\\_RTCEventErrorCallback\(\)](#)

#### 34.2.3 Extension Peripheral Control functions

This subsection provides functions allowing to

- Writes a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Sets the Smooth calibration parameters.
- [HAL\\_RTCEx\\_BKUPWrite\(\)](#)
- [HAL\\_RTCEx\\_BKUPRead\(\)](#)
- [HAL\\_RTCEx\\_SetSmoothCalib\(\)](#)

### 34.2.4 HAL\_RTCEx\_SetTamper

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTamper</b> (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Sets Tamper.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sTamper</b>: Pointer to Tamper Structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• By calling this API we disable the tamper interrupt for all tampers.</li> <li>• Tamper can be enabled only if ASOE and CCO bit are reset</li> </ul>

### 34.2.5 HAL\_RTCEx\_SetTamper\_IT

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT</b> (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Sets Tamper with interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sTamper</b>: Pointer to RTC Tamper.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• By calling this API we force the tamper interrupt for all tampers.</li> <li>• Tamper can be enabled only if ASOE and CCO bit are reset</li> </ul>

### 34.2.6 HAL\_RTCEx\_DeactivateTamper

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper</b> (RTC_HandleTypeDef * hrtc, uint32_t Tamper)
Function Description	Deactivates Tamper.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Tamper</b>: Selected tamper pin. This parameter can be a value of Tamper Pins Definitions</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 34.2.7 HAL\_RTCEx\_TamperIRQHandler

Function Name	<b>void HAL_RTCEx_TamperIRQHandler (RTC_HandleTypeDef * hrtc)</b>
Function Description	This function handles Tamper interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 34.2.8 HAL\_RTCEx\_Tamper1EventCallback

Function Name	<b>void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 34.2.9 HAL\_RTCEx\_PollForTamper1Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	This function handles Tamper1 Polling.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li><li>• <b>Timeout</b>: Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 34.2.10 HAL\_RTCEx\_SetSecond\_IT

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetSecond_IT (RTC_HandleTypeDef * hrtc)</b>
Function Description	Sets Interrupt for second.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 34.2.11 HAL\_RTCEx\_DeactivateSecond

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateSecond</b>
---------------	---

(RTC\_HandleTypeDef \* hrtc)

Function Description	Deactivates Second.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 34.2.12 HAL\_RTCEx\_RTCIRQHandler

Function Name	<b>void HAL_RTCEx_RTCIRQHandler (RTC_HandleTypeDef * hrtc)</b>
Function Description	This function handles second interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 34.2.13 HAL\_RTCEx\_RTCEventCallback

Function Name	<b>void HAL_RTCEx_RTCEventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Second event callback.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 34.2.14 HAL\_RTCEx\_RTCEventErrorCallback

Function Name	<b>void HAL_RTCEx_RTCEventErrorCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Second event error callback.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 34.2.15 HAL\_RTCEx\_BKUPWrite

Function Name	<b>void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)</b>
Function Description	Writes a data in a specified RTC Backup data register.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>BackupRegister</b>: RTC Backup data Register number. This</li> </ul>

parameter can be: RTC\_BKP\_DRx where x can be from 1 to 10 (or 42) to specify the register (depending devices).

- **Data:** Data to be written in the specified RTC Backup data register.

Return values

- None

### 34.2.16 HAL\_RTCEX\_BKUPRead

**Function Name**      **uint32\_t HAL\_RTCEX\_BKUPRead (RTC\_HandleTypeDef \* hrtc, uint32\_t BackupRegister)**

**Function Description**      Reads data from the specified RTC Backup data Register.

- Parameters**
- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
  - **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC\_BKP\_DRx where x can be from 1 to 10 (or 42) to specify the register (depending devices).

Return values

- Read value

### 34.2.17 HAL\_RTCEX\_SetSmoothCalib

**Function Name**      **HAL\_StatusTypeDef HAL\_RTCEX\_SetSmoothCalib (RTC\_HandleTypeDef \* hrtc, uint32\_t SmoothCalibPeriod, uint32\_t SmoothCalibPlusPulses, uint32\_t SmoothCalibMinusPulsesValue)**

**Function Description**      Sets the Smooth calibration parameters.

- Parameters**
- **hrtc:** RTC handle
  - **SmoothCalibPeriod:** Not used (only present for compatibility with another families)
  - **SmoothCalibPlusPulses:** Not used (only present for compatibility with another families)
  - **SmoothCalibMinusPulsesValue:** specifies the RTC Clock Calibration value. This parameter must be a number between 0 and 0x7F.

Return values

- HAL status

## 34.3 RTCEX Firmware driver defines

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

### 34.3.1 RTCEX

RTCEX

*Alias define maintained for legacy*

[HAL\\_RTCEX\\_TamperTimeStampIRQHandler](#)

*Backup Registers Definitions*

RTC\_BKP\_DR1  
RTC\_BKP\_DR2  
RTC\_BKP\_DR3  
RTC\_BKP\_DR4  
RTC\_BKP\_DR5  
RTC\_BKP\_DR6  
RTC\_BKP\_DR7  
RTC\_BKP\_DR8  
RTC\_BKP\_DR9  
RTC\_BKP\_DR10  
RTC\_BKP\_DR11  
RTC\_BKP\_DR12  
RTC\_BKP\_DR13  
RTC\_BKP\_DR14  
RTC\_BKP\_DR15  
RTC\_BKP\_DR16  
RTC\_BKP\_DR17  
RTC\_BKP\_DR18  
RTC\_BKP\_DR19  
RTC\_BKP\_DR20  
RTC\_BKP\_DR21  
RTC\_BKP\_DR22  
RTC\_BKP\_DR23  
RTC\_BKP\_DR24  
RTC\_BKP\_DR25  
RTC\_BKP\_DR26  
RTC\_BKP\_DR27  
RTC\_BKP\_DR28  
RTC\_BKP\_DR29  
RTC\_BKP\_DR30  
RTC\_BKP\_DR31  
RTC\_BKP\_DR32  
RTC\_BKP\_DR33  
RTC\_BKP\_DR34  
RTC\_BKP\_DR35  
RTC\_BKP\_DR36

RTC\_BKP\_DR37

RTC\_BKP\_DR38

RTC\_BKP\_DR39

RTC\_BKP\_DR40

RTC\_BKP\_DR41

RTC\_BKP\_DR42

### ***RTCEX Exported Macros***

`__HAL_RTC_TAMPER_ENABLE_IT`

#### **Description:**

- Enable the RTC Tamper interrupt.

#### **Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
  - `RTC_IT_TAMP1`: Tamper A interrupt

#### **Return value:**

- None:

`__HAL_RTC_TAMPER_DISABLE_IT`

#### **Description:**

- Disable the RTC Tamper interrupt.

#### **Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
  - `RTC_IT_TAMP1`: Tamper A interrupt

#### **Return value:**

- None:

`__HAL_RTC_TAMPER_GET_IT_SOURCE`

#### **Description:**

- Check whether the specified RTC Tamper interrupt has been enabled or not.

#### **Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Tamper interrupt sources to be checked. This parameter can be:



– RTC\_IT\_TAMP1

\_\_HAL\_RTC\_TAMPER\_GET\_FLAG

**Return value:**

- None:

**Description:**

- Get the selected RTC Tamper's flag status.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_FLAG\_\_: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  - RTC\_FLAG\_TAMP1F

**Return value:**

- None:

**Description:**

- Get the selected RTC Tamper's flag status.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC Tamper interrupt sources to be checked. This parameter can be:
  - RTC\_IT\_TAMP1

**Return value:**

- None:

\_\_HAL\_RTC\_TAMPER\_GET\_IT

**Description:**

- Clear the RTC Tamper's pending flags.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_FLAG\_\_: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  - RTC\_FLAG\_TAMP1F

**Return value:**

- None:

\_\_HAL\_RTC\_TAMPER\_CLEAR\_FLAG

**Description:**

- Enable the RTC Second interrupt.

**Parameters:**

\_\_HAL\_RTC\_SECOND\_ENABLE\_IT

**\_\_HAL\_RTC\_SECOND\_DISABLE\_IT**

- **\_\_HANDLE\_\_**: specifies the RTC handle.
- **\_\_INTERRUPT\_\_**: specifies the RTC Second interrupt sources to be enabled. This parameter can be any combination of the following values:
  - **RTC\_IT\_SEC**: Second A interrupt

**Return value:**

- None:

**Description:**

- Disable the RTC Second interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the RTC handle.
- **\_\_INTERRUPT\_\_**: specifies the RTC Second interrupt sources to be disabled. This parameter can be any combination of the following values:
  - **RTC\_IT\_SEC**: Second A interrupt

**Return value:**

- None:

**\_\_HAL\_RTC\_SECOND\_GET\_IT\_SOURCE****Description:**

- Check whether the specified RTC Second interrupt has occurred or not.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the RTC handle.
- **\_\_INTERRUPT\_\_**: specifies the RTC Second interrupt sources to be enabled or disabled. This parameter can be:
  - **RTC\_IT\_SEC**: Second A interrupt

**Return value:**

- None:

**\_\_HAL\_RTC\_SECOND\_GET\_FLAG****Description:**

- Get the selected RTC Second's flag status.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the RTC handle.
- **\_\_FLAG\_\_**: specifies the RTC

Second Flag sources to be enabled or disabled. This parameter can be:

- RTC\_FLAG\_SEC

**Return value:**

- None:

**Description:**

- Clear the RTC Second's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Second Flag sources to be enabled or disabled. This parameter can be:
  - RTC\_FLAG\_SEC

**Return value:**

- None:

**Description:**

- Enable the RTC Overflow interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Overflow interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RTC\_IT\_OW: Overflow A interrupt

**Return value:**

- None:

**Description:**

- Disable the RTC Overflow interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Overflow interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RTC\_IT\_OW: Overflow A interrupt

**Return value:**

`__HAL_RTC_SECOND_CLEAR_FLAG`

`__HAL_RTC_OVERFLOW_ENABLE_IT`

`__HAL_RTC_OVERFLOW_DISABLE_IT`

`__HAL_RTC_OVERFLOW_GET_IT_SOURCE`

- None:

**Description:**

- Check whether the specified RTC Overflow interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Overflow interrupt sources to be enabled or disabled. This parameter can be:
  - `RTC_IT_OW`: Overflow A interrupt

**Return value:**

- None:

`__HAL_RTC_OVERFLOW_GET_FLAG`**Description:**

- Get the selected RTC Overflow's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Overflow Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_OW`

**Return value:**

- None:

`__HAL_RTC_OVERFLOW_CLEAR_FLAG`**Description:**

- Clear the RTC Overflow's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Overflow Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_OW`

**Return value:**

- None:

**Private macros to check input parameters**`IS_RTC_TAMPER``IS_RTC_TAMPER_TRIGGER`

IS\_RTC\_BKP

IS\_RTC\_SMOOTH\_CALIB\_MINUS

***Tamper Pins Definitions***

RTC\_TAMPER\_1      Select tamper to be enabled (mainly for legacy purposes)

***Tamper Trigger Definitions***

RTC\_TAMPERTRIGGER\_LOWLEVEL      A high level on the TAMPER pin resets all data backup registers (if TPE bit is set)

RTC\_TAMPERTRIGGER\_HIGHLEVEL      A low level on the TAMPER pin resets all data backup registers (if TPE bit is set)

## 35 HAL SD Generic Driver

### 35.1 SD Firmware driver registers structures

#### 35.1.1 SD\_HandleTypeDef

*SD\_HandleTypeDef* is defined in the stm32f1xx\_hal\_sd.h

##### Data Fields

- *SD\_TypeDef \* Instance*
- *SD\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *uint32\_t CardType*
- *uint32\_t RCA*
- *uint32\_t CSD*
- *uint32\_t CID*
- *\_\_IO uint32\_t SdTransferCplt*
- *\_\_IO uint32\_t SdTransferErr*
- *\_\_IO uint32\_t DmaTransferCplt*
- *\_\_IO uint32\_t SdOperation*
- *DMA\_HandleTypeDef \* hdmarx*
- *DMA\_HandleTypeDef \* hdmatx*

##### Field Documentation

- *SD\_TypeDef\* SD\_HandleTypeDef::Instance* SDIO register base address
- *SD\_InitTypeDef SD\_HandleTypeDef::Init* SD required parameters
- *HAL\_LockTypeDef SD\_HandleTypeDef::Lock* SD locking object
- *uint32\_t SD\_HandleTypeDef::CardType* SD card type
- *uint32\_t SD\_HandleTypeDef::RCA* SD relative card address
- *uint32\_t SD\_HandleTypeDef::CSD[4]* SD card specific data table
- *uint32\_t SD\_HandleTypeDef::CID[4]* SD card identification number table
- *\_\_IO uint32\_t SD\_HandleTypeDef::SdTransferCplt* SD transfer complete flag in non blocking mode
- *\_\_IO uint32\_t SD\_HandleTypeDef::SdTransferErr* SD transfer error flag in non blocking mode
- *\_\_IO uint32\_t SD\_HandleTypeDef::DmaTransferCplt* SD DMA transfer complete flag
- *\_\_IO uint32\_t SD\_HandleTypeDef::SdOperation* SD transfer operation (read/write)
- *DMA\_HandleTypeDef\* SD\_HandleTypeDef::hdmarx* SD Rx DMA handle parameters
- *DMA\_HandleTypeDef\* SD\_HandleTypeDef::hdmatx* SD Tx DMA handle parameters

#### 35.1.2 HAL\_SD\_CSDTypeDef

*HAL\_SD\_CSDTypeDef* is defined in the stm32f1xx\_hal\_sd.h

##### Data Fields

- `__IO uint8_t CSDStruct`
- `__IO uint8_t SysSpecVersion`
- `__IO uint8_t Reserved1`
- `__IO uint8_t TAAC`
- `__IO uint8_t NSAC`
- `__IO uint8_t MaxBusClkFrec`
- `__IO uint16_t CardComdClasses`
- `__IO uint8_t RdBlockLen`
- `__IO uint8_t PartBlockRead`
- `__IO uint8_t WrBlockMisalign`
- `__IO uint8_t RdBlockMisalign`
- `__IO uint8_t DSRImpl`
- `__IO uint8_t Reserved2`
- `__IO uint32_t DeviceSize`
- `__IO uint8_t MaxRdCurrentVDDMin`
- `__IO uint8_t MaxRdCurrentVDDMax`
- `__IO uint8_t MaxWrCurrentVDDMin`
- `__IO uint8_t MaxWrCurrentVDDMax`
- `__IO uint8_t DeviceSizeMul`
- `__IO uint8_t EraseGrSize`
- `__IO uint8_t EraseGrMul`
- `__IO uint8_t WrProtectGrSize`
- `__IO uint8_t WrProtectGrEnable`
- `__IO uint8_t ManDefIECC`
- `__IO uint8_t WrSpeedFact`
- `__IO uint8_t MaxWrBlockLen`
- `__IO uint8_t WriteBlockPaPartial`
- `__IO uint8_t Reserved3`
- `__IO uint8_t ContentProtectAppli`
- `__IO uint8_t FileFormatGrouop`
- `__IO uint8_t CopyFlag`
- `__IO uint8_t PermWrProtect`
- `__IO uint8_t TempWrProtect`
- `__IO uint8_t FileFormat`
- `__IO uint8_t ECC`
- `__IO uint8_t CSD_CRC`
- `__IO uint8_t Reserved4`

#### Field Documentation

- `__IO uint8_t HAL_SD_CSDTypeDef::CSDStruct` CSD structure
- `__IO uint8_t HAL_SD_CSDTypeDef::SysSpecVersion` System specification version
- `__IO uint8_t HAL_SD_CSDTypeDef::Reserved1` Reserved
- `__IO uint8_t HAL_SD_CSDTypeDef::TAAC` Data read access time 1
- `__IO uint8_t HAL_SD_CSDTypeDef::NSAC` Data read access time 2 in CLK cycles
- `__IO uint8_t HAL_SD_CSDTypeDef::MaxBusClkFrec` Max. bus clock frequency
- `__IO uint16_t HAL_SD_CSDTypeDef::CardComdClasses` Card command classes
- `__IO uint8_t HAL_SD_CSDTypeDef::RdBlockLen` Max. read data block length
- `__IO uint8_t HAL_SD_CSDTypeDef::PartBlockRead` Partial blocks for read allowed
- `__IO uint8_t HAL_SD_CSDTypeDef::WrBlockMisalign` Write block misalignment
- `__IO uint8_t HAL_SD_CSDTypeDef::RdBlockMisalign` Read block misalignment
- `__IO uint8_t HAL_SD_CSDTypeDef::DSRImpl` DSR implemented

- `__IO uint8_t HAL_SD_CSDTypeDef::Reserved2` Reserved
- `__IO uint32_t HAL_SD_CSDTypeDef::DeviceSize` Device Size
- `__IO uint8_t HAL_SD_CSDTypeDef::MaxRdCurrentVDDMin` Max. read current @ VDD min
- `__IO uint8_t HAL_SD_CSDTypeDef::MaxRdCurrentVDDMax` Max. read current @ VDD max
- `__IO uint8_t HAL_SD_CSDTypeDef::MaxWrCurrentVDDMin` Max. write current @ VDD min
- `__IO uint8_t HAL_SD_CSDTypeDef::MaxWrCurrentVDDMax` Max. write current @ VDD max
- `__IO uint8_t HAL_SD_CSDTypeDef::DeviceSizeMul` Device size multiplier
- `__IO uint8_t HAL_SD_CSDTypeDef::EraseGrSize` Erase group size
- `__IO uint8_t HAL_SD_CSDTypeDef::EraseGrMul` Erase group size multiplier
- `__IO uint8_t HAL_SD_CSDTypeDef::WrProtectGrSize` Write protect group size
- `__IO uint8_t HAL_SD_CSDTypeDef::WrProtectGrEnable` Write protect group enable
- `__IO uint8_t HAL_SD_CSDTypeDef::ManDeflECC` Manufacturer default ECC
- `__IO uint8_t HAL_SD_CSDTypeDef::WrSpeedFact` Write speed factor
- `__IO uint8_t HAL_SD_CSDTypeDef::MaxWrBlockLen` Max. write data block length
- `__IO uint8_t HAL_SD_CSDTypeDef::WriteBlockPaPartial` Partial blocks for write allowed
- `__IO uint8_t HAL_SD_CSDTypeDef::Reserved3` Reserved
- `__IO uint8_t HAL_SD_CSDTypeDef::ContentProtectAppli` Content protection application
- `__IO uint8_t HAL_SD_CSDTypeDef::FileFormatGroup` File format group
- `__IO uint8_t HAL_SD_CSDTypeDef::CopyFlag` Copy flag (OTP)
- `__IO uint8_t HAL_SD_CSDTypeDef::PermWrProtect` Permanent write protection
- `__IO uint8_t HAL_SD_CSDTypeDef::TempWrProtect` Temporary write protection
- `__IO uint8_t HAL_SD_CSDTypeDef::FileFormat` File format
- `__IO uint8_t HAL_SD_CSDTypeDef::ECC` ECC code
- `__IO uint8_t HAL_SD_CSDTypeDef::CSD_CRC` CSD CRC
- `__IO uint8_t HAL_SD_CSDTypeDef::Reserved4` Always 1

### 35.1.3 HAL\_SD\_CIDTypeDef

`HAL_SD_CIDTypeDef` is defined in the `stm32f1xx_hal_sd.h`

#### Data Fields

- `__IO uint8_t ManufacturerID`
- `__IO uint16_t OEM_AppliID`
- `__IO uint32_t ProdName1`
- `__IO uint8_t ProdName2`
- `__IO uint8_t ProdRev`
- `__IO uint32_t ProdSN`
- `__IO uint8_t Reserved1`
- `__IO uint16_t ManufactDate`
- `__IO uint8_t CID_CRC`
- `__IO uint8_t Reserved2`

#### Field Documentation

- `__IO uint8_t HAL_SD_CIDTypeDef::ManufacturerID` Manufacturer ID



- `__IO uint16_t HAL_SD_CIDTypeDef::OEM_ApplID` OEM/Application ID
- `__IO uint32_t HAL_SD_CIDTypeDef::ProdName1` Product Name part1
- `__IO uint8_t HAL_SD_CIDTypeDef::ProdName2` Product Name part2
- `__IO uint8_t HAL_SD_CIDTypeDef::ProdRev` Product Revision
- `__IO uint32_t HAL_SD_CIDTypeDef::ProdSN` Product Serial Number
- `__IO uint8_t HAL_SD_CIDTypeDef::Reserved1` Reserved1
- `__IO uint16_t HAL_SD_CIDTypeDef::ManufactDate` Manufacturing Date
- `__IO uint8_t HAL_SD_CIDTypeDef::CID_CRC` CID CRC
- `__IO uint8_t HAL_SD_CIDTypeDef::Reserved2` Always 1

### 35.1.4 HAL\_SD\_CardStatusTypeDef

*HAL\_SD\_CardStatusTypeDef* is defined in the stm32f1xx\_hal\_sd.h

#### Data Fields

- `__IO uint8_t DAT_BUS_WIDTH`
- `__IO uint8_t SECURED_MODE`
- `__IO uint16_t SD_CARD_TYPE`
- `__IO uint32_t SIZE_OF_PROTECTED_AREA`
- `__IO uint8_t SPEED_CLASS`
- `__IO uint8_t PERFORMANCE_MOVE`
- `__IO uint8_t AU_SIZE`
- `__IO uint16_t ERASE_SIZE`
- `__IO uint8_t ERASE_TIMEOUT`
- `__IO uint8_t ERASE_OFFSET`

#### Field Documentation

- `__IO uint8_t HAL_SD_CardStatusTypeDef::DAT_BUS_WIDTH` Shows the currently defined data bus width
- `__IO uint8_t HAL_SD_CardStatusTypeDef::SECURED_MODE` Card is in secured mode of operation
- `__IO uint16_t HAL_SD_CardStatusTypeDef::SD_CARD_TYPE` Carries information about card type
- `__IO uint32_t HAL_SD_CardStatusTypeDef::SIZE_OF_PROTECTED_AREA` Carries information about the capacity of protected area
- `__IO uint8_t HAL_SD_CardStatusTypeDef::SPEED_CLASS` Carries information about the speed class of the card
- `__IO uint8_t HAL_SD_CardStatusTypeDef::PERFORMANCE_MOVE` Carries information about the card's performance move
- `__IO uint8_t HAL_SD_CardStatusTypeDef::AU_SIZE` Carries information about the card's allocation unit size
- `__IO uint16_t HAL_SD_CardStatusTypeDef::ERASE_SIZE` Determines the number of AUs to be erased in one operation
- `__IO uint8_t HAL_SD_CardStatusTypeDef::ERASE_TIMEOUT` Determines the timeout for any number of AU erase
- `__IO uint8_t HAL_SD_CardStatusTypeDef::ERASE_OFFSET` Carries information about the erase offset

### 35.1.5 HAL\_SD\_CardInfoTypeDef

*HAL\_SD\_CardInfoTypeDef* is defined in the stm32f1xx\_hal\_sd.h

#### Data Fields

- *HAL\_SD\_CSDTypeDef* *SD\_csd*
- *HAL\_SD\_CIDTypeDef* *SD\_cid*
- *uint64\_t* *CardCapacity*
- *uint32\_t* *CardBlockSize*
- *uint16\_t* *RCA*
- *uint8\_t* *CardType*

#### Field Documentation

- *HAL\_SD\_CSDTypeDef* *HAL\_SD\_CardInfoTypeDef::SD\_csd* SD card specific data register
- *HAL\_SD\_CIDTypeDef* *HAL\_SD\_CardInfoTypeDef::SD\_cid* SD card identification number register
- *uint64\_t* *HAL\_SD\_CardInfoTypeDef::CardCapacity* Card capacity
- *uint32\_t* *HAL\_SD\_CardInfoTypeDef::CardBlockSize* Card block size
- *uint16\_t* *HAL\_SD\_CardInfoTypeDef::RCA* SD relative card address
- *uint8\_t* *HAL\_SD\_CardInfoTypeDef::CardType* SD card type

## 35.2 SD Firmware driver API description

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

### 35.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDIO and GPIO) are performed by the user in *HAL\_SD\_MspInit()* function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDIO memories which uses the HAL SDIO driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDIO low level resources by implement the *HAL\_SD\_MspInit()* API:
  - a. Enable the SDIO interface clock using *\_\_HAL\_RCC\_SDIO\_CLK\_ENABLE()*;
  - b. SDIO pins configuration for SD card
    - Enable the clock for the SDIO GPIOs using the functions *\_\_HAL\_RCC\_GPIOx\_CLK\_ENABLE()*;
    - Configure these SDIO pins as alternate function pull-up using *HAL\_GPIO\_Init()* and according to your pin assignment;
  - c. DMA Configuration if you need to use DMA process (*HAL\_SD\_ReadBlocks\_DMA()* and *HAL\_SD\_WriteBlocks\_DMA()* APIs).
    - Enable the DMAx interface clock using *\_\_HAL\_RCC\_DMAx\_CLK\_ENABLE()*;
    - Configure the DMA using the function *HAL\_DMA\_Init()* with predeclared and filled.
  - d. NVIC configuration if you need to use interrupt process when using DMA transfer.

- Configure the SDIO and DMA interrupt priorities using functions `HAL_NVIC_SetPriority()`; DMA priority is superior to SDIO's priority
  - Enable the NVIC DMA and SDIO IRQs using function `HAL_NVIC_EnableIRQ()`
  - SDIO interrupts are managed using the macros `__HAL_SD_SDIO_ENABLE_IT()` and `__HAL_SD_SDIO_DISABLE_IT()` inside the communication process.
  - SDIO interrupts pending bits are managed using the macros `__HAL_SD_SDIO_GET_IT()` and `__HAL_SD_SDIO_CLEAR_IT()`
2. At this stage, you can perform SD read/write/erase operations after SD card initialization

### SD Card Initialization and configuration

To initialize the SD Card, use the `HAL_SD_Init()` function. It Initializes the SD Card and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Apply the SD Card initialization process at 400KHz and check the SD Card type (Standard Capacity or High Capacity). You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDIO\_CK) is computed as follows:  $SDIO\_CK = SDIOCLK / (ClockDiv + 2)$  In initialization mode and according to the SD Card standard, make sure that the SDIO\_CK frequency doesn't exceed 400KHz.
2. Get the SD CID and CSD data. All these information are managed by the `SDCardInfo` structure. This structure provide also ready computed SD Card capacity and Block size. These information are stored in SD handle structure in case of future use.
3. Configure the SD Card Data transfer frequency. The card transfer frequency is set to  $SDIOCLK / (SDIO\_TRANSFER\_CLK\_DIV + 2)$ . You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDIO\_CK) is computed as follows:  $SDIO\_CK = SDIOCLK / (ClockDiv + 2)$  In transfer mode and according to the SD Card standard, make sure that the SDIO\_CK frequency doesn't exceed 25MHz and 50MHz in High-speed mode switch.
4. Select the corresponding SD Card according to the address read with the step 2.
5. Configure the SD Card in wide bus mode: 4-bits data.

### SD Card Read operation

- You can read from SD card in polling mode by using function `HAL_SD_ReadBlocks()`. This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.
- You can read from SD card in DMA mode by using function `HAL_SD_ReadBlocks_DMA()`. This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to call the function `HAL_SD_CheckReadOperation()`, to insure that the read transfer is done correctly in both DMA and SD sides.

### SD Card Write operation

- You can write to SD card in polling mode by using function `HAL_SD_WriteBlocks()`. This function support only 512-bytes block length (the block size should be chosen as

512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.

- You can write to SD card in DMA mode by using function `HAL_SD_WriteBlocks_DMA()`. This function support only 512-bytes block length (the block size should be chosen as 512 byte). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to call the function `HAL_SD_CheckWriteOperation()`, to insure that the write transfer is done correctly in both DMA and SD sides.

### SD card status

- At any time, you can check the SD Card status and get the SD card state by using the `HAL_SD_GetStatus()` function. This function checks first if the SD card is still connected and then get the internal SD Card transfer state.
- You can also get the SD card SD Status register by using the `HAL_SD_SendSDStatus()` function.

### SD HAL driver macros list

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

- `__HAL_SD_SDIO_ENABLE` : Enable the SD device
- `__HAL_SD_SDIO_DISABLE` : Disable the SD device
- `__HAL_SD_SDIO_DMA_ENABLE`: Enable the SDIO DMA transfer
- `__HAL_SD_SDIO_DMA_DISABLE`: Disable the SDIO DMA transfer
- `__HAL_SD_SDIO_ENABLE_IT`: Enable the SD device interrupt
- `__HAL_SD_SDIO_DISABLE_IT`: Disable the SD device interrupt
- `__HAL_SD_SDIO_GET_FLAG`: Check whether the specified SD flag is set or not
- `__HAL_SD_SDIO_CLEAR_FLAG`: Clear the SD's pending flags You can refer to the SD HAL driver header file for more useful macros

## 35.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the SD card device to be ready for use.

- [\*`HAL\_SD\_Init\(\)`\*](#)
- [\*`HAL\_SD\_DeInit\(\)`\*](#)
- [\*`HAL\_SD\_MspInit\(\)`\*](#)
- [\*`HAL\_SD\_MspDeInit\(\)`\*](#)

## 35.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to SD card.

- [\*`HAL\_SD\_ReadBlocks\(\)`\*](#)
- [\*`HAL\_SD\_WriteBlocks\(\)`\*](#)
- [\*`HAL\_SD\_ReadBlocks\_DMA\(\)`\*](#)
- [\*`HAL\_SD\_WriteBlocks\_DMA\(\)`\*](#)
- [\*`HAL\_SD\_CheckReadOperation\(\)`\*](#)
- [\*`HAL\_SD\_CheckWriteOperation\(\)`\*](#)
- [\*`HAL\_SD\_Erase\(\)`\*](#)

- [HAL\\_SD\\_IRQHandler\(\)](#)
- [HAL\\_SD\\_XferCpltCallback\(\)](#)
- [HAL\\_SD\\_XferErrorCallback\(\)](#)
- [HAL\\_SD\\_DMA\\_RxCpltCallback\(\)](#)
- [HAL\\_SD\\_DMA\\_RxErrorCallback\(\)](#)
- [HAL\\_SD\\_DMA\\_TxCpltCallback\(\)](#)
- [HAL\\_SD\\_DMA\\_TxErrorCallback\(\)](#)

### 35.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the SD card operations.

- [HAL\\_SD\\_Get\\_CardInfo\(\)](#)
- [HAL\\_SD\\_WideBusOperation\\_Config\(\)](#)
- [HAL\\_SD\\_StopTransfer\(\)](#)
- [HAL\\_SD\\_HighSpeed\(\)](#)

### 35.2.5 Peripheral State functions

This subsection permits to get in runtime the status of the peripheral and the data flow.

- [HAL\\_SD\\_SendSDStatus\(\)](#)
- [HAL\\_SD\\_GetStatus\(\)](#)
- [HAL\\_SD\\_GetCardStatus\(\)](#)

### 35.2.6 HAL\_SD\_Init

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * SDCardInfo)</b>
Function Description	Initializes the SD card according to the specified parameters in the SD_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> <li>• <b>SDCardInfo</b>: HAL_SD_CardInfoTypeDef structure for SD card information</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL SD error state</li> </ul>

### 35.2.7 HAL\_SD\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)</b>
Function Description	De-Initializes the SD card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 35.2.8 HAL\_SD\_MspInit

Function Name	<b>void HAL_SD_MspInit (SD_HandleTypeDef * hsd)</b>
Function Description	Initializes the SD MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 35.2.9 HAL\_SD\_MspDeInit

Function Name	<b>void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)</b>
Function Description	De-Initialize SD MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 35.2.10 HAL\_SD\_ReadBlocks

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)</b>
Function Description	Reads block(s) from a specified address in a card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> <li>• <b>pReadBuffer</b>: pointer to the buffer that will contain the received data</li> <li>• <b>ReadAddr</b>: Address from where data is to be read</li> <li>• <b>BlockSize</b>: SD card Data block size (in bytes) This parameter should be 512</li> <li>• <b>NumberOfBlocks</b>: Number of SD blocks to read</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

### 35.2.11 HAL\_SD\_WriteBlocks

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)</b>
Function Description	Allows to write block(s) to a specified address in a card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> <li>• <b>pWriteBuffer</b>: pointer to the buffer that will contain the data to transmit</li> <li>• <b>WriteAddr</b>: Address from where data is to be written</li> <li>• <b>BlockSize</b>: SD card Data block size (in bytes) This parameter should be 512.</li> <li>• <b>NumberOfBlocks</b>: Number of SD blocks to write</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

### 35.2.12 HAL\_SD\_ReadBlocks\_DMA

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_ReadBlocks_DMA</b> (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)
Function Description	Reads block(s) from a specified address in a card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> <li>• <b>pReadBuffer</b>: Pointer to the buffer that will contain the received data</li> <li>• <b>ReadAddr</b>: Address from where data is to be read</li> <li>• <b>BlockSize</b>: SD card Data block size</li> <li>• <b>NumberOfBlocks</b>: Number of blocks to read.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API should be followed by the function HAL_SD_CheckReadOperation() to check the completion of the read process</li> <li>• BlockSize must be 512 bytes.</li> </ul>

### 35.2.13 HAL\_SD\_WriteBlocks\_DMA

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_WriteBlocks_DMA</b> (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)
Function Description	Writes block(s) to a specified address in a card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> <li>• <b>pWriteBuffer</b>: pointer to the buffer that will contain the data to transmit</li> <li>• <b>WriteAddr</b>: Address from where data is to be read</li> <li>• <b>BlockSize</b>: the SD card Data block size</li> <li>• <b>NumberOfBlocks</b>: Number of blocks to write</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API should be followed by the function HAL_SD_CheckWriteOperation() to check the completion of the write process (by SD current status polling).</li> <li>• BlockSize must be 512 bytes.</li> </ul>

### 35.2.14 HAL\_SD\_CheckReadOperation

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_CheckReadOperation</b> (SD_HandleTypeDef * hsd, uint32_t Timeout)
Function Description	This function waits until the SD DMA data read transfer is finished.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

### 35.2.15 HAL\_SD\_CheckWriteOperation

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_CheckWriteOperation (SD_HandleTypeDef * hsd, uint32_t Timeout)</b>
Function Description	This function waits until the SD DMA data write transfer is finished.
Parameters	<ul style="list-style-type: none"><li>• <b>hsd</b>: SD handle</li><li>• <b>Timeout</b>: Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• SD Card error state</li></ul>

### 35.2.16 HAL\_SD\_Erase

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint64_t Startaddr, uint64_t Endaddr)</b>
Function Description	Erases the specified memory area of the given SD card.
Parameters	<ul style="list-style-type: none"><li>• <b>hsd</b>: SD handle</li><li>• <b>Startaddr</b>: Start byte address</li><li>• <b>Endaddr</b>: End byte address</li></ul>
Return values	<ul style="list-style-type: none"><li>• SD Card error state</li></ul>

### 35.2.17 HAL\_SD\_IRQHandler

Function Name	<b>void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)</b>
Function Description	This function handles SD card interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hsd</b>: SD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 35.2.18 HAL\_SD\_XferCpltCallback

Function Name	<b>void HAL_SD_XferCpltCallback (SD_HandleTypeDef * hsd)</b>
Function Description	SD end of transfer callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hsd</b>: SD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 35.2.19 HAL\_SD\_XferErrorCallback

Function Name	<b>void HAL_SD_XferErrorCallback (SD_HandleTypeDef * hsd)</b>
Function Description	SD Transfer Error callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hsd</b>: SD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>



### 35.2.20 HAL\_SD\_DMA\_RxCpltCallback

Function Name	<b>void HAL_SD_DMA_RxCpltCallback (DMA_HandleTypeDef * hdma)</b>
Function Description	SD Transfer complete Rx callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma</b>: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 35.2.21 HAL\_SD\_DMA\_RxErrorCallback

Function Name	<b>void HAL_SD_DMA_RxErrorCallback (DMA_HandleTypeDef * hdma)</b>
Function Description	SD DMA transfer complete Rx error callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma</b>: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 35.2.22 HAL\_SD\_DMA\_TxCpltCallback

Function Name	<b>void HAL_SD_DMA_TxCpltCallback (DMA_HandleTypeDef * hdma)</b>
Function Description	SD Transfer complete Tx callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma</b>: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 35.2.23 HAL\_SD\_DMA\_TxErrorCallback

Function Name	<b>void HAL_SD_DMA_TxErrorCallback (DMA_HandleTypeDef * hdma)</b>
Function Description	SD DMA transfer complete error Tx callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma</b>: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 35.2.24 HAL\_SD\_Get\_CardInfo

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_Get_CardInfo</b> (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * pCardInfo)
Function Description	Returns information about specific card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> <li>• <b>pCardInfo</b>: Pointer to a HAL_SD_CardInfoTypeDef structure that contains all SD card information</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

### 35.2.25 HAL\_SD\_WideBusOperation\_Config

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_WideBusOperation_Config</b> (SD_HandleTypeDef * hsd, uint32_t WideMode)
Function Description	Enables wide bus operation for the requested card if supported by card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> <li>• <b>WideMode</b>: Specifies the SD card wide bus mode This parameter can be one of the following values: SDIO_BUS_WIDE_8B: 8-bit data transfer (Only for MMC) SDIO_BUS_WIDE_4B: 4-bit data transfer SDIO_BUS_WIDE_1B: 1-bit data transfer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

### 35.2.26 HAL\_SD\_StopTransfer

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_StopTransfer</b> (SD_HandleTypeDef * hsd)
Function Description	Aborts an ongoing data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

### 35.2.27 HAL\_SD\_HighSpeed

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_HighSpeed</b> (SD_HandleTypeDef * hsd)
Function Description	Switches the SD card to High Speed mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This operation should be followed by the configuration of PLL to have SDIOCK clock between 67 and 75 MHz</li> </ul>

**35.2.28 HAL\_SD\_SendSDStatus**

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_SendSDStatus</b> (SD_HandleTypeDef * hsd, uint32_t * pSDstatus)
Function Description	Returns the current SD card's status.
Parameters	<ul style="list-style-type: none"> <li><b>hsd</b>: SD handle</li> <li><b>pSDstatus</b>: Pointer to the buffer that will contain the SD card status SD Status register)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>SD Card error state</li> </ul>

**35.2.29 HAL\_SD\_GetStatus**

Function Name	<b>HAL_SD_TransferStateTypeDef HAL_SD_GetStatus</b> (SD_HandleTypeDef * hsd)
Function Description	Gets the current sd card data status.
Parameters	<ul style="list-style-type: none"> <li><b>hsd</b>: SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>Data Transfer state</li> </ul>

**35.2.30 HAL\_SD\_GetCardStatus**

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_GetCardStatus</b> (SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypeDef * pCardStatus)
Function Description	Gets the SD card status.
Parameters	<ul style="list-style-type: none"> <li><b>hsd</b>: SD handle</li> <li><b>pCardStatus</b>: Pointer to the HAL_SD_CardStatusTypeDef structure that will contain the SD card status information</li> </ul>
Return values	<ul style="list-style-type: none"> <li>SD Card error state</li> </ul>

**35.3 SD Firmware driver defines**

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

**35.3.1 SD**

SD

***SD Exported Constants***

SD_CMD_GO_IDLE_STATE	Resets the SD memory card.
SD_CMD_SEND_OP_COND	Sends host capacity support information and activates the card's initialization process.

SD_CMD_ALL_SEND_CID	Asks any card connected to the host to send the CID numbers on the CMD line.
SD_CMD_SET_REL_ADDR	Asks the card to publish a new relative address (RCA).
SD_CMD_SET_DSR	Programs the DSR of all cards.
SD_CMD_SDIO_SEN_OP_COND	Sends host capacity support information (HCS) and asks the accessed card to send its operating condition register (OCR) content in the response on the CMD line.
SD_CMD_HS_SWITCH	Checks switchable function (mode 0) and switch card function (mode 1).
SD_CMD_SEL_DESEL_CARD	Selects the card by its own relative address and gets deselected by any other address
SD_CMD_HS_SEND_EXT_CSD	Sends SD Memory Card interface condition, which includes host supply voltage information and asks the card whether card supports voltage.
SD_CMD_SEND_CSD	Addressed card sends its card specific data (CSD) on the CMD line.
SD_CMD_SEND_CID	Addressed card sends its card identification (CID) on the CMD line.
SD_CMD_READ_DAT_UNTIL_STOP	SD card doesn't support it.
SD_CMD_STOP_TRANSMISSION	Forces the card to stop transmission.
SD_CMD_SEND_STATUS	Addressed card sends its status register.
SD_CMD_HS_BUSTEST_READ	
SD_CMD_GO_INACTIVE_STATE	Sends an addressed card into the inactive state.
SD_CMD_SET_BLOCKLEN	Sets the block length (in bytes for SDSC) for all following block commands (read, write, lock). Default block length is fixed to 512 Bytes. Not effective for SDHS and SDXC.

SD_CMD_READ_SINGLE_BLOCK	Reads single block of size selected by SET_BLOCKLEN in case of SDSC, and a block of fixed 512 bytes in case of SDHC and SDXC.
SD_CMD_READ_MULT_BLOCK	Continuously transfers data blocks from card to host until interrupted by STOP_TRANSMISSION command.
SD_CMD_HS_BUSTEST_WRITE	64 bytes tuning pattern is sent for SDR50 and SDR104.
SD_CMD_WRITE_DAT_UNTIL_STOP	Speed class control command.
SD_CMD_SET_BLOCK_COUNT	Specify block count for CMD18 and CMD25.
SD_CMD_WRITE_SINGLE_BLOCK	Writes single block of size selected by SET_BLOCKLEN in case of SDSC, and a block of fixed 512 bytes in case of SDHC and SDXC.
SD_CMD_WRITE_MULT_BLOCK	Continuously writes blocks of data until a STOP_TRANSMISSION follows.
SD_CMD_PROG_CID	Reserved for manufacturers.
SD_CMD_PROG_CSD	Programming of the programmable bits of the CSD.
SD_CMD_SET_WRITE_PROT	Sets the write protection bit of the addressed group.
SD_CMD_CLR_WRITE_PROT	Clears the write protection bit of the addressed group.
SD_CMD_SEND_WRITE_PROT	Asks the card to send the status of the write protection bits.
SD_CMD_SD_ERASE_GRP_START	Sets the address of the first write block to be erased. (For SD card only).
SD_CMD_SD_ERASE_GRP_END	Sets the address of the last write block of the continuous range to be erased.
SD_CMD_ERASE_GRP_START	Sets the address of the first write block to be erased. Reserved for each

	command system set by switch function command (CMD6).
SD_CMD_ERASE_GRP_END	Sets the address of the last write block of the continuous range to be erased. Reserved for each command system set by switch function command (CMD6).
SD_CMD_ERASE	Reserved for SD security applications.
SD_CMD_FAST_IO	SD card doesn't support it (Reserved).
SD_CMD_GO_IRQ_STATE	SD card doesn't support it (Reserved).
SD_CMD_LOCK_UNLOCK	Sets/resets the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command.
SD_CMD_APP_CMD	Indicates to the card that the next command is an application specific command rather than a standard command.
SD_CMD_GEN_CMD	Used either to transfer a data block to the card or to get a data block from the card for general purpose/application specific commands.
SD_CMD_NO_CMD	
SD_CMD_APP_SD_SET_BUSWIDTH	SDIO_APP_CMD should be sent before sending these commands. (ACMD6) Defines the data bus width to be used for data transfer. The allowed data bus widths are given in SCR register.
SD_CMD_SD_APP_STAUS	(ACMD13) Sends the SD status.
SD_CMD_SD_APP_SEND_NUM_WRITE_BLOCKS	(ACMD22) Sends the number of the written (without errors) write blocks. Responds with 32bit+CRC data block.
SD_CMD_SD_APP_OP_COND	(ACMD41) Sends host capacity support information

	(HCS) and asks the accessed card to send its operating condition register (OCR) content in the response on the CMD line.
SD_CMD_SD_APP_SET_CLR_CARD_DETECT	(ACMD42) Connects/Disconnects the 50 KOhm pull-up resistor on CD/DAT3 (pin 1) of the card.
SD_CMD_SD_APP_SEND_SCR	Reads the SD Configuration Register (SCR).
SD_CMD_SDIO_RW_DIRECT	For SD I/O card only, reserved for security specification.
SD_CMD_SDIO_RW_EXTENDED	For SD I/O card only, reserved for security specification.
SD_CMD_SD_APP_GET_MKB	SD_CMD_APP_CMD should be sent before sending these commands. For SD card only
SD_CMD_SD_APP_GET_MID	For SD card only
SD_CMD_SD_APP_SET_CER_RN1	For SD card only
SD_CMD_SD_APP_GET_CER_RN2	For SD card only
SD_CMD_SD_APP_SET_CER_RES2	For SD card only
SD_CMD_SD_APP_GET_CER_RES1	For SD card only
SD_CMD_SD_APP_SECURE_READ_MULTIPLE_BLOCK	For SD card only
SD_CMD_SD_APP_SECURE_WRITE_MULTIPLE_BLOCK	For SD card only
SD_CMD_SD_APP_SECURE_ERASE	For SD card only
SD_CMD_SD_APP_CHANGE_SECURE_AREA	For SD card only
SD_CMD_SD_APP_SECURE_WRITE_MKB	For SD card only
STD_CAPACITY_SD_CARD_V1_1	
STD_CAPACITY_SD_CARD_V2_0	
HIGH_CAPACITY_SD_CARD	
MULTIMEDIA_CARD	
SECURE_DIGITAL_IO_CARD	
HIGH_SPEED_MULTIMEDIA_CARD	
SECURE_DIGITAL_IO_COMBO_CARD	
HIGH_CAPACITY_MMC_CARD	

**SD Exported Macros**

__HAL_SD_SDIO_ENABLE	<b>Description:</b>
----------------------	---------------------

	<ul style="list-style-type: none"><li>• Enable the SD device.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: SD Handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None:</li></ul>
<code>__HAL_SD_SDIO_DISABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Disable the SD device.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: SD Handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None:</li></ul>
<code>__HAL_SD_SDIO_DMA_ENABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Enable the SDIO DMA transfer.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: SD Handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None:</li></ul>
<code>__HAL_SD_SDIO_DMA_DISABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Disable the SDIO DMA transfer.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: SD Handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None:</li></ul>
<code>__HAL_SD_SDIO_ENABLE_IT</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Enable the SD device interrupt.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: SD Handle</li><li>• <code>__INTERRUPT__</code>: specifies the SDIO interrupt sources to be enabled. This parameter can be one or a combination of the following values:<ul style="list-style-type: none"><li>– <code>SDIO_IT_CCRCFAIL</code>: Command response received (CRC check failed) interrupt</li><li>– <code>SDIO_IT_DCRCFAIL</code>: Data block sent/received (CRC check failed) interrupt</li><li>– <code>SDIO_IT_CTIMEOUT</code>: Command response timeout interrupt</li><li>– <code>SDIO_IT_DTIMEOUT</code>: Data timeout interrupt</li><li>– <code>SDIO_IT_TXUNDERR</code>: Transmit FIFO underrun error interrupt</li><li>– <code>SDIO_IT_RXOVERR</code>: Received FIFO</li></ul></li></ul>



- overrun error interrupt
- SDIO\_IT\_CMDREND: Command response received (CRC check passed) interrupt
- SDIO\_IT\_CMDSENT: Command sent (no response required) interrupt
- SDIO\_IT\_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO\_IT\_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO\_IT\_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO\_IT\_CMDACT: Command transfer in progress interrupt
- SDIO\_IT\_TXACT: Data transmit in progress interrupt
- SDIO\_IT\_RXACT: Data receive in progress interrupt
- SDIO\_IT\_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO\_IT\_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO\_IT\_TXFIFO: Transmit FIFO full interrupt
- SDIO\_IT\_RXFIFO: Receive FIFO full interrupt
- SDIO\_IT\_TXFIFOE: Transmit FIFO empty interrupt
- SDIO\_IT\_RXFIFOE: Receive FIFO empty interrupt
- SDIO\_IT\_TXDAVL: Data available in transmit FIFO interrupt
- SDIO\_IT\_RXDAVL: Data available in receive FIFO interrupt
- SDIO\_IT\_SDIOIT: SD I/O interrupt received interrupt
- SDIO\_IT\_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

**Return value:**

- None:

**\_\_HAL\_SD\_SDIO\_DISABLE\_IT****Description:**

- Disable the SD device interrupt.

**Parameters:**

- \_\_HANDLE\_\_: SD Handle
- \_\_INTERRUPT\_\_: specifies the SDIO interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDIO\_IT\_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO\_IT\_DCRCFAIL: Data block

- sent/received (CRC check failed) interrupt
- SDIO\_IT\_CTIMEOUT: Command response timeout interrupt
- SDIO\_IT\_DTIMEOUT: Data timeout interrupt
- SDIO\_IT\_TXUNDERR: Transmit FIFO underrun error interrupt
- SDIO\_IT\_RXOVERR: Received FIFO overrun error interrupt
- SDIO\_IT\_CMDREND: Command response received (CRC check passed) interrupt
- SDIO\_IT\_CMDSSENT: Command sent (no response required) interrupt
- SDIO\_IT\_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO\_IT\_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO\_IT\_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO\_IT\_CMDACT: Command transfer in progress interrupt
- SDIO\_IT\_TXACT: Data transmit in progress interrupt
- SDIO\_IT\_RXACT: Data receive in progress interrupt
- SDIO\_IT\_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO\_IT\_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO\_IT\_TXFIFOOF: Transmit FIFO full interrupt
- SDIO\_IT\_RXFIFOOF: Receive FIFO full interrupt
- SDIO\_IT\_TXFIFOE: Transmit FIFO empty interrupt
- SDIO\_IT\_RXFIFOE: Receive FIFO empty interrupt
- SDIO\_IT\_TXDAVL: Data available in transmit FIFO interrupt
- SDIO\_IT\_RXDAVL: Data available in receive FIFO interrupt
- SDIO\_IT\_SDIOIT: SD I/O interrupt received interrupt
- SDIO\_IT\_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

**Return value:**

- None:

**\_\_HAL\_SD\_SDIO\_GET\_FLAG****Description:**

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

**Parameters:**

- `__HANDLE__`: SD Handle
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `SDIO_FLAG_CCRCFAIL`: Command response received (CRC check failed)
  - `SDIO_FLAG_DCRCFAIL`: Data block sent/received (CRC check failed)
  - `SDIO_FLAG_CTIMEOUT`: Command response timeout
  - `SDIO_FLAG_DTIMEOUT`: Data timeout
  - `SDIO_FLAG_TXUNDERR`: Transmit FIFO underrun error
  - `SDIO_FLAG_RXOVERR`: Received FIFO overrun error
  - `SDIO_FLAG_CMDREND`: Command response received (CRC check passed)
  - `SDIO_FLAG_CMDSENT`: Command sent (no response required)
  - `SDIO_FLAG_DATAEND`: Data end (data counter, `SDIDCOUNT`, is zero)
  - `SDIO_FLAG_STBITERR`: Start bit not detected on all data signals in wide bus mode.
  - `SDIO_FLAG_DBCKEND`: Data block sent/received (CRC check passed)
  - `SDIO_FLAG_CMDACT`: Command transfer in progress
  - `SDIO_FLAG_TXACT`: Data transmit in progress
  - `SDIO_FLAG_RXACT`: Data receive in progress
  - `SDIO_FLAG_TXFIFOHE`: Transmit FIFO Half Empty
  - `SDIO_FLAG_RXFIFOHF`: Receive FIFO Half Full
  - `SDIO_FLAG_TXFIFO`: Transmit FIFO full
  - `SDIO_FLAG_RXFIFO`: Receive FIFO full
  - `SDIO_FLAG_TXFIFOE`: Transmit FIFO empty
  - `SDIO_FLAG_RXFIFOE`: Receive FIFO empty
  - `SDIO_FLAG_TXDAVL`: Data available in transmit FIFO
  - `SDIO_FLAG_RXDAVL`: Data available in receive FIFO
  - `SDIO_FLAG_SDIOIT`: SD I/O interrupt received
  - `SDIO_FLAG_CEATAEND`: CE-ATA command completion signal received for CMD61

**Return value:**

`__HAL_SD_SDIO_CLEAR_FLAG`

- The: new state of SD FLAG (SET or RESET).

**Description:**

- Clear the SD's pending flags.

**Parameters:**

- `__HANDLE__`: SD Handle
- `__FLAG__`: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - `SDIO_FLAG_CCRCFAIL`: Command response received (CRC check failed)
  - `SDIO_FLAG_DCRCFAIL`: Data block sent/received (CRC check failed)
  - `SDIO_FLAG_CTIMEOUT`: Command response timeout
  - `SDIO_FLAG_DTIMEOUT`: Data timeout
  - `SDIO_FLAG_TXUNDERR`: Transmit FIFO underrun error
  - `SDIO_FLAG_RXOVERR`: Received FIFO overrun error
  - `SDIO_FLAG_CMDREND`: Command response received (CRC check passed)
  - `SDIO_FLAG_CMDSSENT`: Command sent (no response required)
  - `SDIO_FLAG_DATAEND`: Data end (data counter, `SDIDCOUNT`, is zero)
  - `SDIO_FLAG_STBITERR`: Start bit not detected on all data signals in wide bus mode
  - `SDIO_FLAG_DBCKEND`: Data block sent/received (CRC check passed)
  - `SDIO_FLAG_SDIOIT`: SD I/O interrupt received
  - `SDIO_FLAG_CEATAEND`: CE-ATA command completion signal received for CMD61

**Return value:**

- None:

`__HAL_SD_SDIO_GET_IT`**Description:**

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

**Parameters:**

- `__HANDLE__`: SD Handle
- `__INTERRUPT__`: specifies the SDIO interrupt source to check. This parameter can be one of the following values:
  - `SDIO_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
  - `SDIO_IT_DCRCFAIL`: Data block sent/received (CRC check failed) interrupt

- SDIO\_IT\_CTIMEOUT: Command response timeout interrupt
- SDIO\_IT\_DTIMEOUT: Data timeout interrupt
- SDIO\_IT\_TXUNDERR: Transmit FIFO underrun error interrupt
- SDIO\_IT\_RXOVERR: Received FIFO overrun error interrupt
- SDIO\_IT\_CMDREND: Command response received (CRC check passed) interrupt
- SDIO\_IT\_CMDSSENT: Command sent (no response required) interrupt
- SDIO\_IT\_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO\_IT\_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO\_IT\_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO\_IT\_CMDACT: Command transfer in progress interrupt
- SDIO\_IT\_TXACT: Data transmit in progress interrupt
- SDIO\_IT\_RXACT: Data receive in progress interrupt
- SDIO\_IT\_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO\_IT\_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO\_IT\_TXFIFOOF: Transmit FIFO full interrupt
- SDIO\_IT\_RXFIFOOF: Receive FIFO full interrupt
- SDIO\_IT\_TXFIFOE: Transmit FIFO empty interrupt
- SDIO\_IT\_RXFIFOE: Receive FIFO empty interrupt
- SDIO\_IT\_TXDAVL: Data available in transmit FIFO interrupt
- SDIO\_IT\_RXDAVL: Data available in receive FIFO interrupt
- SDIO\_IT\_SDIOIT: SD I/O interrupt received interrupt
- SDIO\_IT\_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

**Return value:**

- The: new state of SD IT (SET or RESET).

**Description:**

- Clear the SD's interrupt pending bits.

**Parameters:**

`__HAL_SD_SDIO_CLEAR_IT`

- `__HANDLE__`: : SD Handle
- `__INTERRUPT__`: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - `SDIO_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
  - `SDIO_IT_DCRCFAIL`: Data block sent/received (CRC check failed) interrupt
  - `SDIO_IT_CTIMEOUT`: Command response timeout interrupt
  - `SDIO_IT_DTIMEOUT`: Data timeout interrupt
  - `SDIO_IT_TXUNDERR`: Transmit FIFO underrun error interrupt
  - `SDIO_IT_RXOVERR`: Received FIFO overrun error interrupt
  - `SDIO_IT_CMDREND`: Command response received (CRC check passed) interrupt
  - `SDIO_IT_CMDSSENT`: Command sent (no response required) interrupt
  - `SDIO_IT_DATAEND`: Data end (data counter, `SDIO_DCOUNT`, is zero) interrupt
  - `SDIO_IT_STBITERR`: Start bit not detected on all data signals in wide bus mode interrupt
  - `SDIO_IT_SDIOIT`: SD I/O interrupt received interrupt
  - `SDIO_IT_CEATAEND`: CE-ATA command completion signal received for CMD61

**Return value:**

- None:

***SD Exported Types***`SD_InitTypeDef``SD_TypeDef`***SD Private Constant***`DATA_BLOCK_SIZE``SDIO_STATIC_FLAGS``SDIO_CMD0TIMEOUT``SD_OCR_ADDR_OUT_OF_RANGE``SD_OCR_ADDR_MISALIGNED``SD_OCR_BLOCK_LEN_ERR``SD_OCR_ERASE_SEQ_ERR``SD_OCR_BAD_ERASE_PARAM``SD_OCR_WRITE_PROT_VIOLATION``SD_OCR_LOCK_UNLOCK_FAILED`

SD\_OCR\_COM\_CRC\_FAILED  
SD\_OCR\_ILLEGAL\_CMD  
SD\_OCR\_CARD\_ECC\_FAILED  
SD\_OCR\_CC\_ERROR  
SD\_OCR\_GENERAL\_UNKNOWN\_ERROR  
SD\_OCR\_STREAM\_READ\_UNDERRUN  
SD\_OCR\_STREAM\_WRITE\_OVERRUN  
SD\_OCR\_CID\_CSD\_OVERWRITE  
SD\_OCR\_WP\_ERASE\_SKIP  
SD\_OCR\_CARD\_ECC\_DISABLED  
SD\_OCR\_ERASE\_RESET  
SD\_OCR\_AKE\_SEQ\_ERROR  
SD\_OCR\_ERRORBITS  
SD\_R6\_GENERAL\_UNKNOWN\_ERROR  
SD\_R6\_ILLEGAL\_CMD  
SD\_R6\_COM\_CRC\_FAILED  
SD\_VOLTAGE\_WINDOW\_SD  
SD\_HIGH\_CAPACITY  
SD\_STD\_CAPACITY  
SD\_CHECK\_PATTERN  
SD\_MAX\_VOLT\_TRIAL  
SD\_ALLZERO  
SD\_WIDE\_BUS\_SUPPORT  
SD\_SINGLE\_BUS\_SUPPORT  
SD\_CARD\_LOCKED  
SD\_DATATIMEOUT  
SD\_0TO7BITS  
SD\_8TO15BITS  
SD\_16TO23BITS  
SD\_24TO31BITS  
SD\_MAX\_DATA\_LENGTH  
SD\_HALFFIFO  
SD\_HALFFIFOBYTES  
SD\_CCCC\_LOCK\_UNLOCK  
SD\_CCCC\_WRITE\_PROT  
SD\_CCCC\_ERASE

---

`SD_SDIO_SEND_IF_COND``SDIO_APP_CMD` should be sent before sending these commands.



## 36 HAL SMARTCARD Generic Driver

### 36.1 SMARTCARD Firmware driver registers structures

#### 36.1.1 SMARTCARD\_InitTypeDef

*SMARTCARD\_InitTypeDef* is defined in the `stm32f1xx_hal_smartcard.h`

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t CLKLastBit*
- *uint32\_t Prescaler*
- *uint32\_t GuardTime*
- *uint32\_t NACKState*

##### Field Documentation

- *uint32\_t SMARTCARD\_InitTypeDef::BaudRate* This member configures the SmartCard communication baud rate. The baud rate is computed using the following formula:
  - IntegerDivider = ((PCLKx) / (16 \* (hsmartcard->Init.BaudRate)))
  - FractionalDivider = ((IntegerDivider - ((uint32\_t) IntegerDivider)) \* 16) + 0.5
- *uint32\_t SMARTCARD\_InitTypeDef::WordLength* Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [SMARTCARD\\_Word\\_Length](#)
- *uint32\_t SMARTCARD\_InitTypeDef::StopBits* Specifies the number of stop bits transmitted. This parameter can be a value of [SMARTCARD\\_Stop\\_Bits](#)
- *uint32\_t SMARTCARD\_InitTypeDef::Parity* Specifies the parity mode. This parameter can be a value of [SMARTCARD\\_Parity](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- *uint32\_t SMARTCARD\_InitTypeDef::Mode* Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [SMARTCARD\\_Mode](#)
- *uint32\_t SMARTCARD\_InitTypeDef::CLKPolarity* Specifies the steady state of the serial clock. This parameter can be a value of [SMARTCARD\\_Clock\\_Polarity](#)
- *uint32\_t SMARTCARD\_InitTypeDef::CLKPhase* Specifies the clock transition on which the bit capture is made. This parameter can be a value of [SMARTCARD\\_Clock\\_Phase](#)
- *uint32\_t SMARTCARD\_InitTypeDef::CLKLastBit* Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of [SMARTCARD\\_Last\\_Bit](#)

- **`uint32_t SMARTCARD_InitTypeDef::Prescaler`** Specifies the SmartCard Prescaler value used for dividing the system clock to provide the smartcard clock This parameter can be a value of [SMARTCARD\\_Prescaler](#)
- **`uint32_t SMARTCARD_InitTypeDef::GuardTime`** Specifies the SmartCard Guard Time value in terms of number of baud clocks The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency
- **`uint32_t SMARTCARD_InitTypeDef::NACKState`** Specifies the SmartCard NACK Transmission state This parameter can be a value of [SMARTCARD\\_NACK\\_State](#)

### 36.1.2 SMARTCARD\_HandleTypeDef

**`SMARTCARD_HandleTypeDef`** is defined in the `stm32f1xx_hal_smartcard.h`

#### Data Fields

- **`USART_TypeDef * Instance`**
- **`SMARTCARD_InitTypeDef Init`**
- **`uint8_t * pTxBuffPtr`**
- **`uint16_t TxXferSize`**
- **`uint16_t TxXferCount`**
- **`uint8_t * pRxBuffPtr`**
- **`uint16_t RxXferSize`**
- **`uint16_t RxXferCount`**
- **`DMA_HandleTypeDef * hdmatx`**
- **`DMA_HandleTypeDef * hdmarx`**
- **`HAL_LockTypeDef Lock`**
- **`__IO HAL_SMARTCARD_StateTypeDef State`**
- **`__IO uint32_t ErrorCode`**

#### Field Documentation

- **`USART_TypeDef* SMARTCARD_HandleTypeDef::Instance`** USART registers base address
- **`SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init`** SmartCard communication parameters
- **`uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr`** Pointer to SmartCard Tx transfer Buffer
- **`uint16_t SMARTCARD_HandleTypeDef::TxXferSize`** SmartCard Tx Transfer size
- **`uint16_t SMARTCARD_HandleTypeDef::TxXferCount`** SmartCard Tx Transfer Counter
- **`uint8_t* SMARTCARD_HandleTypeDef::pRxBuffPtr`** Pointer to SmartCard Rx transfer Buffer
- **`uint16_t SMARTCARD_HandleTypeDef::RxXferSize`** SmartCard Rx Transfer size
- **`uint16_t SMARTCARD_HandleTypeDef::RxXferCount`** SmartCard Rx Transfer Counter
- **`DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmatx`** SmartCard Tx DMA Handle parameters
- **`DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmarx`** SmartCard Rx DMA Handle parameters
- **`HAL_LockTypeDef SMARTCARD_HandleTypeDef::Lock`** Locking object
- **`__IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::State`** SmartCard communication state

- `__IO uint32_t SMARTCARD_HandleTypeDef::ErrorCode` SmartCard Error code

## 36.2 SMARTCARD Firmware driver API description

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

### 36.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD\_HandleTypeDef handle structure.
2. Initialize the SMARTCARD low level resources by implementing the HAL\_SMARTCARD\_MspInit() API:
  - a. Enable the interface clock of the USARTx associated to the SMARTCARD.
  - b. SMARTCARD pins configuration:
    - Enable the clock for the SMARTCARD GPIOs.
    - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
  - c. NVIC configuration if you need to use interrupt process (HAL\_SMARTCARD\_Transmit\_IT() and HAL\_SMARTCARD\_Receive\_IT() APIs):
    - Configure the USARTx interrupt priority.
    - Enable the NVIC USART IRQ handle.
  - d. DMA Configuration if you need to use DMA process (HAL\_SMARTCARD\_Transmit\_DMA() and HAL\_SMARTCARD\_Receive\_DMA() APIs):
    - Declare a DMA handle structure for the Tx/Rx channel.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure with the required Tx/Rx parameters.
    - Configure the DMA Tx/Rx channel.
    - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
    - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the SMARTCARD Init structure.
4. Initialize the SMARTCARD registers by calling the HAL\_SMARTCARD\_Init() API:
  - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL\_SMARTCARD\_MspInit(&hsc) API. The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros \_\_HAL\_SMARTCARD\_ENABLE\_IT() and \_\_HAL\_SMARTCARD\_DISABLE\_IT() inside the transmit and receive process.
5. Three operation modes are available within this driver :

#### Polling mode IO operation

- Send an amount of data in blocking mode using HAL\_SMARTCARD\_Transmit()

- Receive an amount of data in blocking mode using HAL\_SMARTCARD\_Receive()

### Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL\_SMARTCARD\_Transmit\_IT()
- At transmission end of transfer HAL\_SMARTCARD\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_SMARTCARD\_Receive\_IT()
- At reception end of transfer HAL\_SMARTCARD\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_RxCpltCallback
- In case of transfer Error, HAL\_SMARTCARD\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_ErrorCallback

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL\_SMARTCARD\_Transmit\_DMA()
- At transmission end of transfer HAL\_SMARTCARD\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_SMARTCARD\_Receive\_DMA()
- At reception end of transfer HAL\_SMARTCARD\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_RxCpltCallback
- In case of transfer Error, HAL\_SMARTCARD\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_ErrorCallback

### SMARTCARD HAL driver macros list

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

- \_\_HAL\_SMARTCARD\_ENABLE: Enable the SMARTCARD peripheral
- \_\_HAL\_SMARTCARD\_DISABLE: Disable the SMARTCARD peripheral
- \_\_HAL\_SMARTCARD\_GET\_FLAG : Check whether the specified SMARTCARD flag is set or not
- \_\_HAL\_SMARTCARD\_CLEAR\_FLAG : Clear the specified SMARTCARD pending flag
- \_\_HAL\_SMARTCARD\_ENABLE\_IT: Enable the specified SMARTCARD interrupt
- \_\_HAL\_SMARTCARD\_DISABLE\_IT: Disable the specified SMARTCARD interrupt
- \_\_HAL\_SMARTCARD\_GET\_IT\_SOURCE: Check whether the specified SMARTCARD interrupt has occurred or not



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

## 36.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in Smartcard mode.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard.

The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler.

- For the Smartcard mode only these parameters can be configured as follows:
  - Baud Rate
  - Word Length => Should be 9 bits (8 bits + parity)
  - Stop Bit
  - Parity: => Should be enabled (see [Table 19: "Smartcard frame formats"](#) ).
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes
  - Prescaler
  - GuardTime
  - NACKState: The Smartcard NACK state
- Recommended SmartCard interface configuration to get the Answer to Reset from the Card:
  - Word Length = 9 Bits
  - 1.5 Stop Bit
  - Even parity
  - BaudRate = 12096 baud
  - Tx and Rx enabled

**Table 19: Smartcard frame formats**

M bit	PCE bit	Smartcard frame
1	1	SB   8 bit data   PB   STB

Please refer to the ISO 7816-3 specification for more details. -@- It is also possible to choose 0.5 stop bit for receiving but it is recommended to use 1.5 stop bits for both transmitting and receiving to avoid switching between the two configurations.

The HAL\_SMARTCARD\_Init() function follows the USART SmartCard configuration procedure (details for the procedure are available in reference manuals (RM0008 for STM32F10Xxx MCUs and RM0041 for STM32F100xx MCUs)).

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

## 36.2.3 IO operation functions

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

Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as: (+) 8 bits plus parity: where M=1 and PCE=1 in the USART\_CR1 register (+) 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART\_CR2 register.

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, the relevant API's return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_SMARTCARD\_TxCpltCallback(), HAL\_SMARTCARD\_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The HAL\_SMARTCARD\_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode APIs are :
  - HAL\_SMARTCARD\_Transmit()
  - HAL\_SMARTCARD\_Receive()
3. Non Blocking mode APIs with Interrupt are :
  - HAL\_SMARTCARD\_Transmit\_IT()
  - HAL\_SMARTCARD\_Receive\_IT()
  - HAL\_SMARTCARD\_IRQHandler()
4. Non Blocking mode functions with DMA are :
  - HAL\_SMARTCARD\_Transmit\_DMA()
  - HAL\_SMARTCARD\_Receive\_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_SMARTCARD\_TxCpltCallback()
  - HAL\_SMARTCARD\_RxCpltCallback()
  - HAL\_SMARTCARD\_ErrorCallback()

Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as:

- 8 bits plus parity: where M=1 and PCE=1 in the USART\_CR1 register
- 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART\_CR2 register. (#) There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA, the relevant API's return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_SMARTCARD\_TxCpltCallback(), HAL\_SMARTCARD\_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The HAL\_SMARTCARD\_ErrorCallback() user callback will be executed when a communication error is detected. (#) Blocking mode APIs are :
    - HAL\_SMARTCARD\_Transmit()
    - HAL\_SMARTCARD\_Receive()
  - (#) Non Blocking mode APIs with Interrupt are :

- HAL\_SMARTCARD\_Transmit\_IT()
- HAL\_SMARTCARD\_Receive\_IT()
- HAL\_SMARTCARD\_IRQHandler() (#) Non Blocking mode functions with DMA are :
- HAL\_SMARTCARD\_Transmit\_DMA()
- HAL\_SMARTCARD\_Receive\_DMA() (#) A set of Transfer Complete Callbacks are provided in non Blocking mode:
- HAL\_SMARTCARD\_TxCpltCallback()
- HAL\_SMARTCARD\_RxCpltCallback()
- HAL\_SMARTCARD\_ErrorCallback()
- [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\(\)](#)

### 36.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of SmartCard communication process and also return Peripheral Errors occurred during communication process

- HAL\_SMARTCARD\_GetState() API can be helpful to check in run-time the state of the SMARTCARD peripheral.
- HAL\_SMARTCARD\_GetError() check in run-time errors that could be occurred during communication.
- [HAL\\_SMARTCARD\\_GetState\(\)](#)
- [HAL\\_SMARTCARD\\_GetError\(\)](#)

### 36.2.5 HAL\_SMARTCARD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	Initializes the SmartCard mode according to the specified parameters in the SMARTCARD_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.6 HAL\_SMARTCARD\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_DeInit</b>
---------------	---



**(SMARTCARD\_HandleTypeDef \* hsc)**

Function Description	DeInitializes the SMARTCARD peripheral.
Parameters	<ul style="list-style-type: none"> <li><b>hsc:</b> Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**36.2.7 HAL\_SMARTCARD\_Msplnit**

Function Name	<b>void HAL_SMARTCARD_Msplnit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD MSP Init.
Parameters	<ul style="list-style-type: none"> <li><b>hsc:</b> Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**36.2.8 HAL\_SMARTCARD\_MspDeInit**

Function Name	<b>void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li><b>hsc:</b> Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**36.2.9 HAL\_SMARTCARD\_Transmit**

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Sends an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hsc:</b> Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> <li><b>pData:</b> Pointer to data buffer</li> <li><b>Size:</b> Amount of data to be sent</li> <li><b>Timeout:</b> Specify timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**36.2.10 HAL\_SMARTCARD\_Receive**



Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive</b> (SMARTCARD_HandleTypeDef * hsc, 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"> <li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be received</li> <li>• <b>Timeout</b>: Specify timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.11 HAL\_SMARTCARD\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT</b> (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Sends an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.12 HAL\_SMARTCARD\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT</b> (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Receives an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.13 HAL\_SMARTCARD\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA</b> (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Sends an amount of data in non-blocking mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.14 HAL\_SMARTCARD\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the SMARTCARD parity is enabled (PCE = 1) the data received contain the parity bit.</li> </ul>

### 36.2.15 HAL\_SMARTCARD\_IRQHandler

Function Name	<b>void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	This function handles SMARTCARD interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 36.2.16 HAL\_SMARTCARD\_TxCpltCallback

Function Name	<b>void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 36.2.17 HAL\_SMARTCARD\_RxCpltCallback

Function Name	<b>void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 36.2.18 HAL\_SMARTCARD\_ErrorCallback

Function Name	<b>void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD error callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 36.2.19 HAL\_SMARTCARD\_GetState

Function Name	<b>HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	Returns the SMARTCARD state.
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL state</li></ul>

### 36.2.20 HAL\_SMARTCARD\_GetError

Function Name	<b>uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	Return the SMARTCARD error code.
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b>: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• SMARTCARD Error Code</li></ul>

## 36.3 SMARTCARD Firmware driver defines

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

### 36.3.1 SMARTCARD

SMARTCARD

#### **SMARTCARD Clock Phase**

SMARTCARD\_PHASE\_1EDGE

SMARTCARD\_PHASE\_2EDGE

#### **SMARTCARD Clock Polarity**

SMARTCARD\_POLARITY\_LOW

SMARTCARD\_POLARITY\_HIGH

#### **SMARTCARD DMA requests**

SMARTCARD\_DMAREQ\_TX

SMARTCARD\_DMAREQ\_RX

#### **SMARTCARD Error Codes**

HAL_SMARTCARD_ERROR_NONE	No error
HAL_SMARTCARD_ERROR_PE	Parity error
HAL_SMARTCARD_ERROR_NE	Noise error
HAL_SMARTCARD_ERROR_FE	frame error
HAL_SMARTCARD_ERROR_ORE	Overrun error
HAL_SMARTCARD_ERROR_DMA	DMA transfer error

#### **SMARTCARD Exported Macros**

\_\_HAL\_SMARTCARD\_RESET\_HANDLE\_STATE

#### **Description:**

- Reset SMARTCARD handle state.

#### **Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

#### **Return value:**

- None:

\_\_HAL\_SMARTCARD\_FLUSH\_DRREGISTER

#### **Description:**

- Flush the Smartcard DR register.

#### **Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART

availability and x value depending on device).

**Return value:**

- None:

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle.  
SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `SMARTCARD_FLAG_TXE`: Transmit data register empty flag
  - `SMARTCARD_FLAG_TC`: Transmission Complete flag
  - `SMARTCARD_FLAG_RXNE`: Receive data register not empty flag
  - `SMARTCARD_FLAG_IDLE`: Idle Line detection flag
  - `SMARTCARD_FLAG_ORE`: OverRun Error flag
  - `SMARTCARD_FLAG_NE`: Noise Error flag
  - `SMARTCARD_FLAG_FE`: Framing Error flag
  - `SMARTCARD_FLAG_PE`: Parity Error flag

**Return value:**

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

**Description:**

- Clear the specified Smartcard pending flags.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle.  
SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

`__HAL_SMARTCARD_GET_FLAG`

`__HAL_SMARTCARD_CLEAR_FLAG`

- `__FLAG__`: specifies the flag to check. This parameter can be any combination of the following values:
  - `SMARTCARD_FLAG_TC`: Transmission Complete flag.
  - `SMARTCARD_FLAG_RXNE`: Receive data register not empty flag.

**Return value:**

- None:
- None:

`__HAL_SMARTCARD_CLEAR_PEFLLAG`**Description:**

- Clear the SMARTCARD PE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

`__HAL_SMARTCARD_CLEAR_FEFLAG`**Description:**

- Clear the SMARTCARD FE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

`__HAL_SMARTCARD_CLEAR_NEFLAG`**Description:**

- Clear the SMARTCARD NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

---

**\_\_HAL\_SMARTCARD\_CLEAR\_OREFLAG****Description:**

- Clear the SMARTCARD ORE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

**\_\_HAL\_SMARTCARD\_CLEAR\_IDLEFLAG****Description:**

- Clear the SMARTCARD IDLE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

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

- Enable the specified SmartCard interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- **\_\_INTERRUPT\_\_**: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
  - SMARTCARD\_IT\_IDLE: Idle line detection interrupt

- SMARTCARD\_IT\_PE: Parity Error interrupt
- SMARTCARD\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

**Description:**

- Disable the specified SmartCard interrupts.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMARTCARD Handle.  
SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- **\_\_INTERRUPT\_\_**: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
  - SMARTCARD\_IT\_IDLE: Idle line detection interrupt
  - SMARTCARD\_IT\_PE: Parity Error interrupt
  - SMARTCARD\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

`__HAL_SMARTCARD_DISABLE_IT`

`__HAL_SMARTCARD_GET_IT_SOURCE`

**Description:**

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

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMARTCARD Handle.  
SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- **\_\_IT\_\_**: specifies the SMARTCARD



interrupt source to check. This parameter can be one of the following values:

- SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
- SMARTCARD\_IT\_TC: Transmission complete interrupt
- SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
- SMARTCARD\_IT\_IDLE: Idle line detection interrupt
- SMARTCARD\_IT\_ERR: Error interrupt
- SMARTCARD\_IT\_PE: Parity Error interrupt

**Return value:**

- The: new state of \_\_IT\_\_ (TRUE or FALSE).

**Description:**

- Enable the USART associated to the SMARTCARD Handle.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

**Description:**

- Disable the USART associated to the SMARTCARD Handle.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

**Description:**

\_\_HAL\_SMARTCARD\_ENABLE

\_\_HAL\_SMARTCARD\_DISABLE

\_\_HAL\_SMARTCARD\_DMA\_REQUEST\_ENA

## BLE

- Enable the SmartCard DMA request.

**Parameters:**

- `__HANDLE__`: specifies the SmartCard Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__REQUEST__`: specifies the SmartCard DMA request. This parameter can be one of the following values:
  - `SMARTCARD_DMAREQ_TX`: SmartCard DMA transmit request
  - `SMARTCARD_DMAREQ_RX`: SmartCard DMA receive request

**Return value:**

- None:

`__HAL_SMARTCARD_DMA_REQUEST_DISABLE`**Description:**

- Disable the SmartCard DMA request.

**Parameters:**

- `__HANDLE__`: specifies the SmartCard Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__REQUEST__`: specifies the SmartCard DMA request. This parameter can be one of the following values:
  - `SMARTCARD_DMAREQ_TX`: SmartCard DMA transmit request
  - `SMARTCARD_DMAREQ_RX`: SmartCard DMA receive request

**Return value:**

- None:

**SMARTCARD Flags**`SMARTCARD_FLAG_TXE``SMARTCARD_FLAG_TC``SMARTCARD_FLAG_RXNE``SMARTCARD_FLAG_IDLE`

SMARTCARD\_FLAG\_ORE

SMARTCARD\_FLAG\_NE

SMARTCARD\_FLAG\_FE

SMARTCARD\_FLAG\_PE

**SMARTCARD Interrupts Definition**

SMARTCARD\_IT\_PE

SMARTCARD\_IT\_TXE

SMARTCARD\_IT\_TC

SMARTCARD\_IT\_RXNE

SMARTCARD\_IT\_IDLE

SMARTCARD\_IT\_ERR

**SMARTCARD Last Bit**

SMARTCARD\_LASTBIT\_DISABLE

SMARTCARD\_LASTBIT\_ENABLE

**SMARTCARD Mode**

SMARTCARD\_MODE\_RX

SMARTCARD\_MODE\_TX

SMARTCARD\_MODE\_TX\_RX

**SMARTCARD NACK State**

SMARTCARD\_NACK\_ENABLE

SMARTCARD\_NACK\_DISABLE

**SMARTCARD Parity**

SMARTCARD\_PARITY\_EVEN

SMARTCARD\_PARITY\_ODD

**SMARTCARD Prescaler**

SMARTCARD\_PRESCALER\_SYCLK\_DIV2      SYCLK divided by 2

SMARTCARD\_PRESCALER\_SYCLK\_DIV4      SYCLK divided by 4

SMARTCARD\_PRESCALER\_SYCLK\_DIV6      SYCLK divided by 6

SMARTCARD\_PRESCALER\_SYCLK\_DIV8      SYCLK divided by 8

SMARTCARD\_PRESCALER\_SYCLK\_DIV10      SYCLK divided by 10

SMARTCARD\_PRESCALER\_SYCLK\_DIV12      SYCLK divided by 12

SMARTCARD\_PRESCALER\_SYCLK\_DIV14      SYCLK divided by 14

SMARTCARD\_PRESCALER\_SYCLK\_DIV16      SYCLK divided by 16

SMARTCARD\_PRESCALER\_SYCLK\_DIV18      SYCLK divided by 18

SMARTCARD\_PRESCALER\_SYCLK\_DIV20      SYCLK divided by 20

SMARTCARD\_PRESCALER\_SYCLK\_DIV22      SYCLK divided by 22

SMARTCARD_PRESCALER_SYSCLOCK_DIV24	SYSCLOCK divided by 24
SMARTCARD_PRESCALER_SYSCLOCK_DIV26	SYSCLOCK divided by 26
SMARTCARD_PRESCALER_SYSCLOCK_DIV28	SYSCLOCK divided by 28
SMARTCARD_PRESCALER_SYSCLOCK_DIV30	SYSCLOCK divided by 30
SMARTCARD_PRESCALER_SYSCLOCK_DIV32	SYSCLOCK divided by 32
SMARTCARD_PRESCALER_SYSCLOCK_DIV34	SYSCLOCK divided by 34
SMARTCARD_PRESCALER_SYSCLOCK_DIV36	SYSCLOCK divided by 36
SMARTCARD_PRESCALER_SYSCLOCK_DIV38	SYSCLOCK divided by 38
SMARTCARD_PRESCALER_SYSCLOCK_DIV40	SYSCLOCK divided by 40
SMARTCARD_PRESCALER_SYSCLOCK_DIV42	SYSCLOCK divided by 42
SMARTCARD_PRESCALER_SYSCLOCK_DIV44	SYSCLOCK divided by 44
SMARTCARD_PRESCALER_SYSCLOCK_DIV46	SYSCLOCK divided by 46
SMARTCARD_PRESCALER_SYSCLOCK_DIV48	SYSCLOCK divided by 48
SMARTCARD_PRESCALER_SYSCLOCK_DIV50	SYSCLOCK divided by 50
SMARTCARD_PRESCALER_SYSCLOCK_DIV52	SYSCLOCK divided by 52
SMARTCARD_PRESCALER_SYSCLOCK_DIV54	SYSCLOCK divided by 54
SMARTCARD_PRESCALER_SYSCLOCK_DIV56	SYSCLOCK divided by 56
SMARTCARD_PRESCALER_SYSCLOCK_DIV58	SYSCLOCK divided by 58
SMARTCARD_PRESCALER_SYSCLOCK_DIV60	SYSCLOCK divided by 60
SMARTCARD_PRESCALER_SYSCLOCK_DIV62	SYSCLOCK divided by 62

**SMARTCARD Private Macros**

SMARTCARD\_CR1\_REG\_INDEX

SMARTCARD\_CR3\_REG\_INDEX

SMARTCARD\_DIV

SMARTCARD\_DIVMANT

SMARTCARD\_DIVFRAQ

SMARTCARD\_BRR

IS\_SMARTCARD\_BAUDRATE

The maximum Baud Rate is derived from the maximum clock on APB (i.e. 72 MHz) divided by the smallest oversampling used on the USART (i.e. 16)   
**\_\_BAUDRATE\_\_**: Baud rate set by the configuration function. Return : TRUE or FALSE

IS\_SMARTCARD\_WORD\_LENGTH

IS\_SMARTCARD\_STOPBITS

IS\_SMARTCARD\_PARITY

IS\_SMARTCARD\_MODE

IS\_SMARTCARD\_POLARITY

IS\_SMARTCARD\_PHASE

IS\_SMARTCARD\_LASTBIT

IS\_SMARTCARD\_NACK\_STATE

IS\_SMARTCARD\_PRESCALER

SMARTCARD\_IT\_MASK

***SMARTCARD Number of Stop Bits***

SMARTCARD\_STOPBITS\_0\_5

SMARTCARD\_STOPBITS\_1\_5

***SMARTCARD Word Length***

SMARTCARD\_WORDLENGTH\_9B

## 37 HAL SPI Generic Driver

### 37.1 SPI Firmware driver registers structures

#### 37.1.1 SPI\_InitTypeDef

*SPI\_InitTypeDef* is defined in the stm32f1xx\_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](#)  
**Note:** The communication clock is derived from the master clock. The slave clock does not need to be set
- *uint32\_t SPI\_InitTypeDef::FirstBit* Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of [SPI\\_MSB\\_LSB\\_transmission](#)
- *uint32\_t SPI\_InitTypeDef::TIMode* Specifies if the TI mode is enabled or not. This parameter can be a value of [SPI\\_TI\\_mode](#)
- *uint32\_t SPI\_InitTypeDef::CRCCalculation* Specifies if the CRC calculation is enabled or not. This parameter can be a value of [SPI\\_CRC\\_Calculation](#)
- *uint32\_t SPI\_InitTypeDef::CRCPolynomial* Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min\_Data = 0 and Max\_Data = 65535

### 37.1.2 `__SPI_HandleTypeDef`

`__SPI_HandleTypeDef` is defined in the `stm32f1xx_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 uint32_t ErrorCode`

#### Field Documentation

- `SPI_TypeDef* __SPI_HandleTypeDef::Instance` SPI registers base address
- `SPI_InitTypeDef __SPI_HandleTypeDef::Init` SPI communication parameters
- `uint8_t* __SPI_HandleTypeDef::pTxBuffPtr` Pointer to SPI Tx transfer Buffer
- `uint16_t __SPI_HandleTypeDef::TxXferSize` SPI Tx transfer size
- `uint16_t __SPI_HandleTypeDef::TxXferCount` SPI Tx Transfer Counter
- `uint8_t* __SPI_HandleTypeDef::pRxBuffPtr` Pointer to SPI Rx transfer Buffer
- `uint16_t __SPI_HandleTypeDef::RxXferSize` SPI Rx transfer size
- `uint16_t __SPI_HandleTypeDef::RxXferCount` SPI Rx Transfer Counter
- `DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx` SPI Tx DMA handle parameters
- `DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx` SPI Rx DMA handle parameters
- `void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)` function pointer on Rx ISR
- `void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)` function pointer on Tx ISR
- `HAL_LockTypeDef __SPI_HandleTypeDef::Lock` SPI locking object
- `__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State` SPI communication state
- `__IO uint32_t __SPI_HandleTypeDef::ErrorCode` SPI Error code

## 37.2 SPI Firmware driver API description

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

### 37.2.1 How to use this driver



The SPI HAL driver can be used as follows:

1. Declare a SPI\_HandleTypeDef handle structure, for example: SPI\_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL\_SPI\_MspInit ()API:
  - a. Enable the SPIx interface clock
  - b. SPI pins configuration
    - Enable the clock for the SPI GPIOs
    - Configure these SPI pins as alternate function push-pull
  - c. NVIC configuration if you need to use interrupt process
    - Configure the SPIx interrupt priority
    - Enable the NVIC SPI IRQ handle
  - d. DMA Configuration if you need to use DMA process
    - Declare a DMA\_HandleTypeDef handle structure for the transmit or receive Channel
    - Enable the DMAx clock
    - Configure the DMA handle parameters
    - Configure the DMA Tx or Rx Channel
    - Associate the initialized hdma\_tx(or \_rx) handle to the hspi DMA Tx (or Rx) handle
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Channel
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() API.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
  - a. Master 2Lines RxOnly
  - b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled
3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL\_SPI\_DMAPause()/ HAL\_SPI\_DMAStop() only under the SPI callbacks

Using the HAL it is not possible to reach all supported SPI frequency with the different SPI Modes. [Table 20: "Maximum SPI frequency for 8-bit SPI data transfers"](#) and [Table 21: "Maximum SPI frequency for 16-bit SPI data transfers"](#) summarize the maximum SPI frequency reached with data size 8bits/16bits, according to frequency used on APBx Peripheral Clock (fPCLK) used by the SPI instance.



The max SPI frequency depend on SPI data size (8bits, 16bits), SPI mode(2 Lines full duplex, 2 lines RxOnly, 1 line TX/RX) and Process mode (Polling, IT, DMA).



1. TX/RX processes are HAL\_SPI\_TransmitReceive(), HAL\_SPI\_TransmitReceive\_IT() and HAL\_SPI\_TransmitReceive\_DMA()
2. RX processes are HAL\_SPI\_Receive(), HAL\_SPI\_Receive\_IT() and HAL\_SPI\_Receive\_DMA()
3. TX processes are HAL\_SPI\_Transmit(), HAL\_SPI\_Transmit\_IT() and HAL\_SPI\_Transmit\_DMA()



Table 20: Maximum SPI frequency for 8-bit SPI data transfers

Process	Transfer mode	2 lines, full duplex		2 line, Rx only		1 line	
		Master	Slave	Master	Slave	Master	Slave
Tx/Rx	Polling	$f_{CPU}/8$	$f_{CPU}/8$	NA	NA	NA	NA
	Interrupt	$f_{CPU}/32$	$f_{CPU}/32$	NA	NA	NA	NA
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	NA	NA
Rx	Polling	$f_{CPU}/4$	$f_{CPU}/8$	$f_{CPU}/128$	$f_{CPU}/16$	$f_{CPU}/128$	$f_{CPU}/8$
	Interrupt	$f_{CPU}/32$	$f_{CPU}/16$	$f_{CPU}/128$	$f_{CPU}/16$	$f_{CPU}/128$	$f_{CPU}/16$
	DMA	$f_{CPU}/2$	$f_{CPU}/2$	$f_{CPU}/128$	$f_{CPU}/16$	$f_{CPU}/128$	$f_{CPU}/2$
Tx	Polling	$f_{CPU}/4$	$f_{CPU}/4$	NA	NA	$f_{CPU}/4$	$f_{CPU}/64$
	Interrupt	$f_{CPU}/8$	$f_{CPU}/16$	NA	NA	$f_{CPU}/8$	$f_{CPU}/128$
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	$f_{CPU}/2$	$f_{CPU}/64$

Table 21: Maximum SPI frequency for 16-bit SPI data transfers

Process	Transfer mode	2 lines, full duplex		2 line, Rx only		1 line	
		Master	Slave	Master	Slave	Master	Slave
Tx/Rx	Polling	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	NA	NA
	Interrupt	$f_{CPU}/16$	$f_{CPU}/16$	NA	NA	NA	NA
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	NA	NA
Rx	Polling	$f_{CPU}/2$	$f_{CPU}/4$	$f_{CPU}/64$	$f_{CPU}/8$	$f_{CPU}/64$	$f_{CPU}/4$
	Interrupt	$f_{CPU}/16$	$f_{CPU}/8$	$f_{CPU}/128$	$f_{CPU}/8$	$f_{CPU}/128$	$f_{CPU}/8$
	DMA	$f_{CPU}/2$	$f_{CPU}/2$	$f_{CPU}/128$	$f_{CPU}/8$	$f_{CPU}/128$	$f_{CPU}/2$
Tx	Polling	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	$f_{CPU}/2$	$f_{CPU}/64$
	Interrupt	$f_{CPU}/4$	$f_{CPU}/8$	NA	NA	$f_{CPU}/4$	$f_{CPU}/256$
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	$f_{CPU}/2$	$f_{CPU}/32$

### 37.2.2 Initialization and de-initialization functions

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

- User must implement HAL\_SPI\_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function HAL\_SPI\_Init() to configure the selected device with the selected configuration:
  - Mode
  - Direction
  - Data Size
  - Clock Polarity and Phase
  - NSS Management
  - BaudRate Prescaler
  - FirstBit
  - TIMode

- CRC Calculation
- CRC Polynomial if CRC enabled
- Call the function HAL\_SPI\_DeInit() to restore the default configuration of the selected SPIx peripheral.
- [HAL\\_SPI\\_Init\(\)](#)
- [HAL\\_SPI\\_DeInit\(\)](#)
- [HAL\\_SPI\\_MspInit\(\)](#)
- [HAL\\_SPI\\_MspDeInit\(\)](#)

### 37.2.3 IO operation functions

The SPI supports master and slave mode :

1. There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_SPI\_TxCpltCallback(), HAL\_SPI\_RxCpltCallback() and HAL\_SPI\_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL\_SPI\_ErrorCallback() user callback will be executed when a communication error is detected
2. APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.
  - [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\\_DMAPause\(\)](#)
  - [HAL\\_SPI\\_DMAResume\(\)](#)
  - [HAL\\_SPI\\_DMAStop\(\)](#)
  - [HAL\\_SPI\\_IRQHandler\(\)](#)
  - [HAL\\_SPI\\_TxCpltCallback\(\)](#)
  - [HAL\\_SPI\\_RxCpltCallback\(\)](#)
  - [HAL\\_SPI\\_TxRxCpltCallback\(\)](#)
  - [HAL\\_SPI\\_TxHalfCpltCallback\(\)](#)
  - [HAL\\_SPI\\_RxHalfCpltCallback\(\)](#)
  - [HAL\\_SPI\\_TxRxHalfCpltCallback\(\)](#)
  - [HAL\\_SPI\\_ErrorCallback\(\)](#)

### 37.2.4 Peripheral State and Errors functions

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

- HAL\_SPI\_GetState() API can be helpful to check in run-time the state of the SPI peripheral
- HAL\_SPI\_GetError() check in run-time Errors occurring during communication
- [HAL\\_SPI\\_GetState\(\)](#)
- [HAL\\_SPI\\_GetError\(\)](#)

### 37.2.5 HAL\_SPI\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)</b>
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"> <li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 37.2.6 HAL\_SPI\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)</b>
Function Description	DeInitializes the SPI peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 37.2.7 HAL\_SPI\_MspInit

Function Name	<b>void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)</b>
Function Description	SPI MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 37.2.8 HAL\_SPI\_MspDeInit

Function Name	<b>void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)</b>
Function Description	SPI MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 37.2.9 HAL\_SPI\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li><li>• <b>pData</b>: pointer to data buffer</li><li>• <b>Size</b>: amount of data to be sent</li><li>• <b>Timeout</b>: Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 37.2.10 HAL\_SPI\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li><li>• <b>pData</b>: pointer to data buffer</li><li>• <b>Size</b>: amount of data to be sent</li><li>• <b>Timeout</b>: Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 37.2.11 HAL\_SPI\_TransmitReceive

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmit and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li><li>• <b>pTxData</b>: pointer to transmission data buffer</li><li>• <b>pRxData</b>: pointer to reception data buffer to be</li><li>• <b>Size</b>: amount of data to be sent</li><li>• <b>Timeout</b>: Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 37.2.12 HAL\_SPI\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmit an amount of data in no-blocking mode with Interrupt.

Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData</b>: pointer to data buffer</li> <li>• <b>Size</b>: amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 37.2.13 HAL\_SPI\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive_IT</b> (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"> <li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData</b>: pointer to data buffer</li> <li>• <b>Size</b>: amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 37.2.14 HAL\_SPI\_TransmitReceive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT</b> (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Transmit and Receive an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pTxData</b>: pointer to transmission data buffer</li> <li>• <b>pRxData</b>: pointer to reception data buffer to be</li> <li>• <b>Size</b>: amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 37.2.15 HAL\_SPI\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit_DMA</b> (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData</b>: pointer to data buffer</li> <li>• <b>Size</b>: amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 37.2.16 HAL\_SPI\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive_DMA</b> (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"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li><li>• <b>pData</b>: pointer to data buffer</li><li>• <b>Size</b>: amount of data to be sent</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>
Notes	<ul style="list-style-type: none"><li>• When the CRC feature is enabled the pData Length must be Size + 1.</li></ul>

### 37.2.17 HAL\_SPI\_TransmitReceive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA</b> (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"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li><li>• <b>pTxData</b>: pointer to transmission data buffer</li><li>• <b>pRxData</b>: pointer to reception data buffer</li><li>• <b>Size</b>: amount of data to be sent</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>
Notes	<ul style="list-style-type: none"><li>• When the CRC feature is enabled the pRxData Length must be Size + 1</li></ul>

### 37.2.18 HAL\_SPI\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DMAPause</b> (SPI_HandleTypeDef * hspi)
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 37.2.19 HAL\_SPI\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DMAResume</b> (SPI_HandleTypeDef * hspi)
---------------	--

Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li><b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 37.2.20 HAL\_SPI\_DMASStop

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DMASStop (SPI_HandleTypeDef * hspi)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li><b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 37.2.21 HAL\_SPI\_IRQHandler

Function Name	<b>void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)</b>
Function Description	This function handles SPI interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 37.2.22 HAL\_SPI\_TxCpltCallback

Function Name	<b>void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li><b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 37.2.23 HAL\_SPI\_RxCpltCallback

Function Name	<b>void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li><b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 37.2.24 HAL\_SPI\_TxRxCpltCallback

---

Function Name	<b>void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 37.2.25 HAL\_SPI\_TxHalfCpltCallback

Function Name	<b>void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 37.2.26 HAL\_SPI\_RxHalfCpltCallback

Function Name	<b>void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 37.2.27 HAL\_SPI\_TxRxHalfCpltCallback

Function Name	<b>void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 37.2.28 HAL\_SPI\_ErrorCallback

Function Name	<b>void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	SPI error callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b>: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>



**37.2.29 HAL\_SPI\_GetState**

Function Name	<b>HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)</b>
Function Description	Return the SPI state.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>SPI state</li> </ul>

**37.2.30 HAL\_SPI\_GetError**

Function Name	<b>uint32_t HAL_SPI_GetError (SPI_HandleTypeDef * hspi)</b>
Function Description	Return the SPI error code.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>SPI Error Code</li> </ul>

**37.3 SPI Firmware driver defines**

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

**37.3.1 SPI**

SPI

***SPI BaudRate Prescaler***

SPI\_BAUDRATEPRESCALER\_2

SPI\_BAUDRATEPRESCALER\_4

SPI\_BAUDRATEPRESCALER\_8

SPI\_BAUDRATEPRESCALER\_16

SPI\_BAUDRATEPRESCALER\_32

SPI\_BAUDRATEPRESCALER\_64

SPI\_BAUDRATEPRESCALER\_128

SPI\_BAUDRATEPRESCALER\_256

***SPI Clock Phase***

SPI\_PHASE\_1EDGE

SPI\_PHASE\_2EDGE

***SPI Clock Polarity***

SPI\_POLARITY\_LOW

SPI\_POLARITY\_HIGH

***SPI CRC Calculation***

SPI\_CRCCALCULATION\_DISABLE

SPI\_CRCCALCULATION\_ENABLE

### ***SPI data size***

SPI\_DATASIZE\_8BIT

SPI\_DATASIZE\_16BIT

### ***SPI Direction mode***

SPI\_DIRECTION\_2LINES

SPI\_DIRECTION\_2LINES\_RXONLY

SPI\_DIRECTION\_1LINE

### ***SPI Error Codes***

HAL\_SPI\_ERROR\_NONE    No error

HAL\_SPI\_ERROR\_MODF    MODF error

HAL\_SPI\_ERROR\_CRC    CRC error

HAL\_SPI\_ERROR\_OVR    OVR error

HAL\_SPI\_ERROR\_DMA    DMA transfer error

HAL\_SPI\_ERROR\_FLAG    Flag: RXNE, TXE, BSY

### ***SPI Exported Macros***

**\_\_HAL\_SPI\_RESET\_HANDLE\_STATE**    **Description:**

- Reset SPI handle state.

#### **Parameters:**

- **\_\_HANDLE\_\_**: specifies the SPI handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

#### **Return value:**

- None:

**\_\_HAL\_SPI\_ENABLE\_IT**

#### **Description:**

- Enable the specified SPI interrupts.

#### **Parameters:**

- **\_\_HANDLE\_\_**: specifies the SPI handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- **\_\_INTERRUPT\_\_**: specifies the interrupt source to enable. This parameter can be one of the following values:
  - SPI\_IT\_TXE: Tx buffer empty interrupt enable
  - SPI\_IT\_RXNE: RX buffer not empty interrupt enable
  - SPI\_IT\_ERR: Error interrupt enable

#### **Return value:**

`__HAL_SPI_DISABLE_IT`

- None:

**Description:**

- Disable the specified SPI interrupts.

**Parameters:**

- `__HANDLE__`: specifies the SPI handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- `__INTERRUPT__`: specifies the interrupt source to disable. This parameter can be one of the following values:
  - `SPI_IT_TXE`: Tx buffer empty interrupt enable
  - `SPI_IT_RXNE`: RX buffer not empty interrupt enable
  - `SPI_IT_ERR`: Error interrupt enable

**Return value:**

- None:

`__HAL_SPI_GET_IT_SOURCE`**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the SPI handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- `__INTERRUPT__`: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - `SPI_IT_TXE`: Tx buffer empty interrupt enable
  - `SPI_IT_RXNE`: RX buffer not empty interrupt enable
  - `SPI_IT_ERR`: Error interrupt enable

**Return value:**

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

`__HAL_SPI_GET_FLAG`**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the SPI handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `SPI_FLAG_RXNE`: Receive buffer not empty flag

- SPI\_FLAG\_TXE: Transmit buffer empty flag
- SPI\_FLAG\_CRCERR: CRC error flag
- SPI\_FLAG\_MODF: Mode fault flag
- SPI\_FLAG\_OVR: Overrun flag
- SPI\_FLAG\_BSY: Busy flag

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

**\_\_HAL\_SPI\_CLEAR\_CRCERRFLAG****Description:**

- Clear the SPI CRCERR pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None:

**\_\_HAL\_SPI\_CLEAR\_MODFFLAG****Description:**

- Clear the SPI MODF pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None:

**\_\_HAL\_SPI\_CLEAR\_OVRFLAG****Description:**

- Clear the SPI OVR pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None:

**\_\_HAL\_SPI\_ENABLE****Description:**

- Enables the SPI.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

\_\_HAL\_SPI\_DISABLE

- None:

**Description:**

- Disables the SPI.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None:

***SPI Flag definition***

SPI\_FLAG\_RXNE

SPI\_FLAG\_TXE

SPI\_FLAG\_CRCERR

SPI\_FLAG\_MODF

SPI\_FLAG\_OVR

SPI\_FLAG\_BSY

***SPI Interrupt configuration definition***

SPI\_IT\_TXE

SPI\_IT\_RXNE

SPI\_IT\_ERR

***SPI mode***

SPI\_MODE\_SLAVE

SPI\_MODE\_MASTER

***SPI MSB LSB transmission***

SPI\_FIRSTBIT\_MSB

SPI\_FIRSTBIT\_LSB

***SPI Private Constants***

SPI\_TIMEOUT\_VALUE

SPI\_INVALID\_CRC\_ERROR

SPI\_VALID\_CRC\_ERROR

***SPI Private Macros***

IS\_SPI\_MODE

**Description:**

- Checks if SPI Mode parameter is in allowed range.

**Parameters:**

- \_\_MODE\_\_: specifies the SPI Mode. This parameter can be a value of

**IS\_SPI\_DIRECTION\_MODE****Return value:**

- None:

**Description:**

- Checks if SPI Direction Mode parameter is in allowed range.

**Parameters:**

- `__MODE__`: specifies the SPI Direction Mode. This parameter can be a value of

**Return value:**

- None:

**IS\_SPI\_DIRECTION\_2LINES\_OR\_1LINE****Description:**

- Checks if SPI Direction Mode parameter is 1 or 2 lines.

**Parameters:**

- `__MODE__`: specifies the SPI Direction Mode.

**Return value:**

- None:

**IS\_SPI\_DIRECTION\_2LINES****Description:**

- Checks if SPI Direction Mode parameter is 2 lines.

**Parameters:**

- `__MODE__`: specifies the SPI Direction Mode.

**Return value:**

- None:

**IS\_SPI\_DATASIZE****Description:**

- Checks if SPI Data Size parameter is in allowed range.

**Parameters:**

- `__DATASIZE__`: specifies the SPI Data Size. This parameter can be a value of

**Return value:**

- None:

**IS\_SPI\_CPOL****Description:**

- Checks if SPI Serial clock steady state parameter is in allowed range.

**Parameters:**

- `__CPOL__`: specifies the SPI serial clock steady state. This parameter can be a

IS\_SPI\_CPHA

value of

**Return value:**

- None:

**Description:**

- Checks if SPI Clock Phase parameter is in allowed range.

**Parameters:**

- `__CPHA__`: specifies the SPI Clock Phase. This parameter can be a value of

**Return value:**

- None:

**Description:**

- Checks if SPI Slave select parameter is in allowed range.

**Parameters:**

- `__NSS__`: specifies the SPI Slave Select management parameter. This parameter can be a value of

**Return value:**

- None:

IS\_SPI\_BAUDRATE\_PRESCALER

**Description:**

- Checks if SPI Baudrate prescaler parameter is in allowed range.

**Parameters:**

- `__PRESCALER__`: specifies the SPI Baudrate prescaler. This parameter can be a value of

**Return value:**

- None:

IS\_SPI\_FIRST\_BIT

**Description:**

- Checks if SPI MSB LSB transmission parameter is in allowed range.

**Parameters:**

- `__BIT__`: specifies the SPI MSB LSB transmission (whether data transfer starts from MSB or LSB bit). This parameter can be a value of

**Return value:**

- None:

IS\_SPI\_TIMODE

**Description:**

## IS\_SPI\_CRC\_CALCULATION

- Checks if SPI TI mode parameter is in allowed range.

**Parameters:**

- `__MODE__`: specifies the SPI TI mode. This parameter can be a value of

**Return value:**

- None:

**Description:**

- Checks if SPI CRC calculation enabled state is in allowed range.

**Parameters:**

- `__CALCULATION__`: specifies the SPI CRC calculation enable state. This parameter can be a value of

**Return value:**

- None:

**Description:**

- Checks if SPI polynomial value to be used for the CRC calculation, is in allowed range.

**Parameters:**

- `__POLYNOMIAL__`: specifies the SPI polynomial value to be used for the CRC calculation. This parameter must be a number between Min\_Data = 0 and Max\_Data = 65535

**Return value:**

- None:

**Description:**

- Sets the SPI transmit-only mode.

**Parameters:**

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None:

**Description:**

- Sets the SPI receive-only mode.

**Parameters:**

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2,

## IS\_SPI\_CRC\_POLYNOMIAL

## SPI\_1LINE\_TX

## SPI\_1LINE\_RX



or 3 to select the SPI peripheral.

**Return value:**

- None:

**Description:**

- Resets the CRC calculation of the SPI.

**Parameters:**

- `__HANDLE__`: specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None:

`SPI_RESET_CRC`

***SPI Slave Select management***

`SPI_NSS_SOFT`

`SPI_NSS_HARD_INPUT`

`SPI_NSS_HARD_OUTPUT`

***SPI TI mode disable***

`SPI_TIMODE_DISABLE`

## 38 HAL SRAM Generic Driver

### 38.1 SRAM Firmware driver registers structures

#### 38.1.1 SRAM\_HandleTypeDef

*SRAM\_HandleTypeDef* is defined in the stm32f1xx\_hal\_sram.h

##### Data Fields

- *FSMC\_NORSRAM\_TypeDef \* Instance*
- *FSMC\_NORSRAM\_EXTENDED\_TypeDef \* Extended*
- *FSMC\_NORSRAM\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_SRAM\_StateTypeDef State*
- *DMA\_HandleTypeDef \* hdma*

##### Field Documentation

- *FSMC\_NORSRAM\_TypeDef\* SRAM\_HandleTypeDef::Instance* Register base address
- *FSMC\_NORSRAM\_EXTENDED\_TypeDef\* SRAM\_HandleTypeDef::Extended* Extended mode register base address
- *FSMC\_NORSRAM\_InitTypeDef SRAM\_HandleTypeDef::Init* SRAM device control configuration parameters
- *HAL\_LockTypeDef SRAM\_HandleTypeDef::Lock* SRAM locking object
- *\_\_IO HAL\_SRAM\_StateTypeDef SRAM\_HandleTypeDef::State* SRAM device access state
- *DMA\_HandleTypeDef\* SRAM\_HandleTypeDef::hdma* Pointer DMA handler

### 38.2 SRAM Firmware driver API description

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

#### 38.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FSMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FSMC to interface with SRAM/PSRAM memories:

1. Declare a *SRAM\_HandleTypeDef* handle structure, for example:  
*SRAM\_HandleTypeDef* hsram; and:
  - Fill the *SRAM\_HandleTypeDef* handle "Init" field with the allowed values of the structure member.
  - Fill the *SRAM\_HandleTypeDef* handle "Instance" field with a predefined base register instance for NOR or SRAM device
  - Fill the *SRAM\_HandleTypeDef* handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode

2. Declare two `FSMC_NORSRAM_TimingTypeDef` structures, for both normal and extended mode timings; for example: `FSMC_NORSRAM_TimingTypeDef Timing` and `FSMC_NORSRAM_TimingTypeDef ExTiming`; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function `HAL_SRAM_Init()`. This function performs the following sequence:
  - a. MSP hardware layer configuration using the function `HAL_SRAM_MspInit()`
  - b. Control register configuration using the FSMC NORSRAM interface function `FSMC_NORSRAM_Init()`
  - c. Timing register configuration using the FSMC NORSRAM interface function `FSMC_NORSRAM_Timing_Init()`
  - d. Extended mode Timing register configuration using the FSMC NORSRAM interface function `FSMC_NORSRAM_Extended_Timing_Init()`
  - e. Enable the SRAM device using the macro `__FSMC_NORSRAM_ENABLE()`
4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
  - `HAL_SRAM_Read()/HAL_SRAM_Write()` for polling read/write access
  - `HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA()` for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs `HAL_SRAM_WriteOperation_Enable()/HAL_SRAM_WriteOperation_Disable()` to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function `HAL_SRAM_GetState()`

### 38.2.2 SRAM Initialization and de\_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

- [`HAL\_SRAM\_Init\(\)`](#)
- [`HAL\_SRAM\_DeInit\(\)`](#)
- [`HAL\_SRAM\_MspInit\(\)`](#)
- [`HAL\_SRAM\_MspDeInit\(\)`](#)
- [`HAL\_SRAM\_DMA\_XferCpltCallback\(\)`](#)
- [`HAL\_SRAM\_DMA\_XferErrorCallback\(\)`](#)

### 38.2.3 SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

- [`HAL\_SRAM\_Read\_8b\(\)`](#)
- [`HAL\_SRAM\_Write\_8b\(\)`](#)
- [`HAL\_SRAM\_Read\_16b\(\)`](#)
- [`HAL\_SRAM\_Write\_16b\(\)`](#)
- [`HAL\_SRAM\_Read\_32b\(\)`](#)
- [`HAL\_SRAM\_Write\_32b\(\)`](#)
- [`HAL\_SRAM\_Read\_DMA\(\)`](#)
- [`HAL\_SRAM\_Write\_DMA\(\)`](#)

### 38.2.4 SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.

- [HAL\\_SRAM\\_WriteOperation\\_Enable\(\)](#)
- [HAL\\_SRAM\\_WriteOperation\\_Disable\(\)](#)

### 38.2.5 SRAM State functions

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

- [HAL\\_SRAM\\_GetState\(\)](#)

### 38.2.6 HAL\_SRAM\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FSMC_NORSRAM_TimingTypeDef * Timing, FSMC_NORSRAM_TimingTypeDef * ExtTiming)</b>
Function Description	Performs the SRAM device initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>Timing</b>: Pointer to SRAM control timing structure</li> <li>• <b>ExtTiming</b>: Pointer to SRAM extended mode timing structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 38.2.7 HAL\_SRAM\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)</b>
Function Description	Performs the SRAM device De-initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 38.2.8 HAL\_SRAM\_Msplnit

Function Name	<b>void HAL_SRAM_Msplnit (SRAM_HandleTypeDef * hsram)</b>
Function Description	SRAM MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 38.2.9 HAL\_SRAM\_MspDeInit

Function Name	<b>void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)</b>
Function Description	SRAM MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 38.2.10 HAL\_SRAM\_DMA\_XferCpltCallback

Function Name	<b>void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)</b>
Function Description	DMA transfer complete callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 38.2.11 HAL\_SRAM\_DMA\_XferErrorCallback

Function Name	<b>void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)</b>
Function Description	DMA transfer complete error callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 38.2.12 HAL\_SRAM\_Read\_8b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)</b>
Function Description	Reads 8-bit buffer from SRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b>: Pointer to read start address</li> <li>• <b>pDstBuffer</b>: Pointer to destination buffer</li> <li>• <b>BufferSize</b>: Size of the buffer to read from memory</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 38.2.13 HAL\_SRAM\_Write\_8b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)</b>
---------------	---

Function Description	Writes 8-bit buffer to SRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b>: Pointer to write start address</li> <li>• <b>pSrcBuffer</b>: Pointer to source buffer to write</li> <li>• <b>BufferSize</b>: Size of the buffer to write to memory</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 38.2.14 HAL\_SRAM\_Read\_16b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Read_16b</b> (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)
Function Description	Reads 16-bit buffer from SRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b>: Pointer to read start address</li> <li>• <b>pDstBuffer</b>: Pointer to destination buffer</li> <li>• <b>BufferSize</b>: Size of the buffer to read from memory</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 38.2.15 HAL\_SRAM\_Write\_16b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Write_16b</b> (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)
Function Description	Writes 16-bit buffer to SRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b>: Pointer to write start address</li> <li>• <b>pSrcBuffer</b>: Pointer to source buffer to write</li> <li>• <b>BufferSize</b>: Size of the buffer to write to memory</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 38.2.16 HAL\_SRAM\_Read\_32b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Read_32b</b> (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)
Function Description	Reads 32-bit buffer from SRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b>: Pointer to read start address</li> <li>• <b>pDstBuffer</b>: Pointer to destination buffer</li> </ul>

- **BufferSize:** Size of the buffer to read from memory
- Return values
  - HAL status

### 38.2.17 HAL\_SRAM\_Write\_32b

- Function Name** `HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)`
- Function Description** Writes 32-bit buffer to SRAM memory.
- Parameters**
- **hsram:** pointer to a SRAM\_HandleTypeDef structure that contains the configuration information for SRAM module.
  - **pAddress:** Pointer to write start address
  - **pSrcBuffer:** Pointer to source buffer to write
  - **BufferSize:** Size of the buffer to write to memory
- Return values**
- HAL status

### 38.2.18 HAL\_SRAM\_Read\_DMA

- Function Name** `HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)`
- Function Description** Reads a Words data from the SRAM memory using DMA transfer.
- Parameters**
- **hsram:** pointer to a SRAM\_HandleTypeDef structure that contains the configuration information for SRAM module.
  - **pAddress:** Pointer to read start address
  - **pDstBuffer:** Pointer to destination buffer
  - **BufferSize:** Size of the buffer to read from memory
- Return values**
- HAL status

### 38.2.19 HAL\_SRAM\_Write\_DMA

- Function Name** `HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)`
- Function Description** Writes a Words data buffer to SRAM memory using DMA transfer.
- Parameters**
- **hsram:** pointer to a SRAM\_HandleTypeDef structure that contains the configuration information for SRAM module.
  - **pAddress:** Pointer to write start address
  - **pSrcBuffer:** Pointer to source buffer to write
  - **BufferSize:** Size of the buffer to write to memory
- Return values**
- HAL status

### 38.2.20 HAL\_SRAM\_WriteOperation\_Enable

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)</b>
Function Description	Enables dynamically SRAM write operation.
Parameters	<ul style="list-style-type: none"><li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 38.2.21 HAL\_SRAM\_WriteOperation\_Disable

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)</b>
Function Description	Disables dynamically SRAM write operation.
Parameters	<ul style="list-style-type: none"><li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 38.2.22 HAL\_SRAM\_GetState

Function Name	<b>HAL_SRAM_StateTypeDef HAL_SRAM_GetState (SRAM_HandleTypeDef * hsram)</b>
Function Description	Returns the SRAM controller state.
Parameters	<ul style="list-style-type: none"><li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL state</li></ul>

## 38.3 SRAM Firmware driver defines

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

### 38.3.1 SRAM

SRAM

#### *SRAM Exported Macros*

<b>__HAL_SRAM_RESET_HANDLE_STATE</b>	<b>Description:</b> <ul style="list-style-type: none"><li>• Reset SRAM handle state.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <b>__HANDLE__</b>: SRAM handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None:</li></ul>
--------------------------------------	---



## 39 HAL TIM Generic Driver

### 39.1 TIM Firmware driver registers structures

#### 39.1.1 TIM\_Base\_InitTypeDef

**TIM\_Base\_InitTypeDef** is defined in the stm32f1xx\_hal\_tim.h

##### Data Fields

- **uint32\_t Prescaler**
- **uint32\_t CounterMode**
- **uint32\_t Period**
- **uint32\_t ClockDivision**
- **uint32\_t RepetitionCounter**

##### Field Documentation

- **uint32\_t TIM\_Base\_InitTypeDef::Prescaler** Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- **uint32\_t TIM\_Base\_InitTypeDef::CounterMode** Specifies the counter mode. This parameter can be a value of [TIM\\_Counter\\_Mode](#)
- **uint32\_t TIM\_Base\_InitTypeDef::Period** Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- **uint32\_t TIM\_Base\_InitTypeDef::ClockDivision** Specifies the clock division. This parameter can be a value of [TIM\\_ClockDivision](#)
- **uint32\_t TIM\_Base\_InitTypeDef::RepetitionCounter** Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned mode
  - the number of half PWM period in center-aligned mode This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.**Note:** This parameter is valid only for TIM1 and TIM8.

#### 39.1.2 TIM\_OC\_InitTypeDef

**TIM\_OC\_InitTypeDef** is defined in the stm32f1xx\_hal\_tim.h

##### Data Fields

- **uint32\_t OCMODE**
- **uint32\_t Pulse**
- **uint32\_t OCPolarity**
- **uint32\_t OCNPolarity**
- **uint32\_t OCFastMode**
- **uint32\_t OCIdleState**
- **uint32\_t OCNIdleState**

**Field Documentation**

- **`uint32_t TIM_OC_InitTypeDef::OCMode`** Specifies the TIM mode. This parameter can be a value of [TIM\\_Output\\_Compare\\_and\\_PWM\\_modes](#)
- **`uint32_t TIM_OC_InitTypeDef::Pulse`** Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between `Min_Data = 0x0000` and `Max_Data = 0xFFFF`
- **`uint32_t TIM_OC_InitTypeDef::OCpolarity`** Specifies the output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_Polarity](#)
- **`uint32_t TIM_OC_InitTypeDef::OCNPolarity`** Specifies the complementary output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_N\\_Polarity](#)  
**Note:**This parameter is valid only for TIM1 and TIM8.
- **`uint32_t TIM_OC_InitTypeDef::OCFastMode`** Specifies the Fast mode state. This parameter can be a value of [TIM\\_Output\\_Fast\\_State](#)  
**Note:**This parameter is valid only in PWM1 and PWM2 mode.
- **`uint32_t TIM_OC_InitTypeDef::OCIdleState`** Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of [TIM\\_Output\\_Compare\\_Idle\\_State](#)  
**Note:**This parameter is valid only for TIM1 and TIM8.
- **`uint32_t TIM_OC_InitTypeDef::OCNIdleState`** Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of [TIM\\_Output\\_Compare\\_N\\_Idle\\_State](#)  
**Note:**This parameter is valid only for TIM1 and TIM8.

**39.1.3 TIM\_OnePulse\_InitTypeDef**

**`TIM_OnePulse_InitTypeDef`** is defined in the `stm32f1xx_hal_tim.h`

**Data Fields**

- **`uint32_t OCMODE`**
- **`uint32_t Pulse`**
- **`uint32_t OCPolarity`**
- **`uint32_t OCNPolarity`**
- **`uint32_t OCIdleState`**
- **`uint32_t OCNIdleState`**
- **`uint32_t ICPolarity`**
- **`uint32_t ICSelection`**
- **`uint32_t ICFILTER`**

**Field Documentation**

- **`uint32_t TIM_OnePulse_InitTypeDef::OCMode`** Specifies the TIM mode. This parameter can be a value of [TIM\\_Output\\_Compare\\_and\\_PWM\\_modes](#)
- **`uint32_t TIM_OnePulse_InitTypeDef::Pulse`** Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between `Min_Data = 0x0000` and `Max_Data = 0xFFFF`
- **`uint32_t TIM_OnePulse_InitTypeDef::OCpolarity`** Specifies the output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_Polarity](#)
- **`uint32_t TIM_OnePulse_InitTypeDef::OCNPolarity`** Specifies the complementary output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_N\\_Polarity](#)  
**Note:**This parameter is valid only for TIM1 and TIM8.
- **`uint32_t TIM_OnePulse_InitTypeDef::OCIdleState`** Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of

***TIM\_Output\_Compare\_Idle\_State***

**Note:** This parameter is valid only for TIM1 and TIM8.

- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OCNIdleState*** Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of

***TIM\_Output\_Compare\_N\_Idle\_State***

**Note:** This parameter is valid only for TIM1 and TIM8.

- ***uint32\_t TIM\_OnePulse\_InitTypeDef::ICPolarity*** Specifies the active edge of the input signal. This parameter can be a value of ***TIM\_Input\_Capture\_Polarity***
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::ICSelection*** Specifies the input. This parameter can be a value of ***TIM\_Input\_Capture\_Selection***
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::ICFilter*** Specifies the input capture filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

### 39.1.4 TIM\_IC\_InitTypeDef

***TIM\_IC\_InitTypeDef*** is defined in the stm32f1xx\_hal\_tim.h

#### Data Fields

- ***uint32\_t ICPolarity***
- ***uint32\_t ICSelection***
- ***uint32\_t ICPrescaler***
- ***uint32\_t ICFilter***

#### Field Documentation

- ***uint32\_t TIM\_IC\_InitTypeDef::ICPolarity*** Specifies the active edge of the input signal. This parameter can be a value of ***TIM\_Input\_Capture\_Polarity***
- ***uint32\_t TIM\_IC\_InitTypeDef::ICSelection*** Specifies the input. This parameter can be a value of ***TIM\_Input\_Capture\_Selection***
- ***uint32\_t TIM\_IC\_InitTypeDef::ICPrescaler*** Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM\_Input\_Capture\_Prescaler***
- ***uint32\_t TIM\_IC\_InitTypeDef::ICFilter*** Specifies the input capture filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

### 39.1.5 TIM\_Encoder\_InitTypeDef

***TIM\_Encoder\_InitTypeDef*** is defined in the stm32f1xx\_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

### 39.1.6 TIM\_ClockConfigTypeDef

**`TIM_ClockConfigTypeDef`** is defined in the `stm32f1xx_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

### 39.1.7 TIM\_ClearInputConfigTypeDef

**`TIM_ClearInputConfigTypeDef`** is defined in the `stm32f1xx_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

**39.1.8 TIM\_SlaveConfigTypeDef**

*TIM\_SlaveConfigTypeDef* is defined in the stm32f1xx\_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

**39.1.9 TIM\_HandleTypeDef**

*TIM\_HandleTypeDef* is defined in the stm32f1xx\_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 [TIM\\_DMA\\_Handle\\_index](#)
- ***HAL\_LockTypeDef TIM\_HandleTypeDef::Lock*** Locking object
- ***\_\_IO HAL\_TIM\_StateTypeDef TIM\_HandleTypeDef::State*** TIM operation state

## 39.2 TIM Firmware driver API description

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

### 39.2.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
  - Input Capture
  - Output Compare
  - PWM generation (Edge and Center-aligned Mode)
  - One-pulse mode output

### 39.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
  - Time Base : `HAL_TIM_Base_MspInit()`
  - Input Capture : `HAL_TIM_IC_MspInit()`
  - Output Compare : `HAL_TIM_OC_MspInit()`
  - PWM generation : `HAL_TIM_PWM_MspInit()`
  - One-pulse mode output : `HAL_TIM_OnePulse_MspInit()`
  - Encoder mode output : `HAL_TIM_Encoder_MspInit()`
2. Initialize the TIM low level resources :
  - a. Enable the TIM interface clock using `__HAL_RCC_TIMx_CLK_ENABLE()`;
  - b. TIM pins configuration
    - Enable the clock for the TIM GPIOs using the following function: `__HAL_GPIOx_CLK_ENABLE()`;
    - Configure these TIM pins in Alternate function mode using `HAL_GPIO_Init()`;
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: `HAL_TIM_ConfigClockSource`, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
  - `HAL_TIM_Base_Init`: to use the Timer to generate a simple time base

- HAL\_TIM\_OC\_Init and HAL\_TIM\_OC\_ConfigChannel: to use the Timer to generate an Output Compare signal.
  - HAL\_TIM\_PWM\_Init and HAL\_TIM\_PWM\_ConfigChannel: to use the Timer to generate a PWM signal.
  - HAL\_TIM\_IC\_Init and HAL\_TIM\_IC\_ConfigChannel: to use the Timer to measure an external signal.
  - HAL\_TIM\_OnePulse\_Init and HAL\_TIM\_OnePulse\_ConfigChannel: to use the Timer in One Pulse Mode.
  - HAL\_TIM\_Encoder\_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
    - Time Base : HAL\_TIM\_Base\_Start(), HAL\_TIM\_Base\_Start\_DMA(), HAL\_TIM\_Base\_Start\_IT()
    - Input Capture : HAL\_TIM\_IC\_Start(), HAL\_TIM\_IC\_Start\_DMA(), HAL\_TIM\_IC\_Start\_IT()
    - Output Compare : HAL\_TIM\_OC\_Start(), HAL\_TIM\_OC\_Start\_DMA(), HAL\_TIM\_OC\_Start\_IT()
    - PWM generation : HAL\_TIM\_PWM\_Start(), HAL\_TIM\_PWM\_Start\_DMA(), HAL\_TIM\_PWM\_Start\_IT()
    - One-pulse mode output : HAL\_TIM\_OnePulse\_Start(), HAL\_TIM\_OnePulse\_Start\_IT()
    - Encoder mode output : HAL\_TIM\_Encoder\_Start(), HAL\_TIM\_Encoder\_Start\_DMA(), HAL\_TIM\_Encoder\_Start\_IT().
  6. The DMA Burst is managed with the two following functions:  
 HAL\_TIM\_DMABurst\_WriteStart() HAL\_TIM\_DMABurst\_ReadStart()

### 39.2.3 Time Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Time Base.
- Stop the Time Base.
- Start the Time Base and enable interrupt.
- Stop the Time Base and disable interrupt.
- Start the Time Base and enable DMA transfer.
- Stop the Time Base and disable DMA transfer.
- [HAL\\_TIM\\_Base\\_Init\(\)](#)
- [HAL\\_TIM\\_Base\\_DeInit\(\)](#)
- [HAL\\_TIM\\_Base\\_MspInit\(\)](#)
- [HAL\\_TIM\\_Base\\_MspDeInit\(\)](#)
- [HAL\\_TIM\\_Base\\_Start\(\)](#)
- [HAL\\_TIM\\_Base\\_Stop\(\)](#)
- [HAL\\_TIM\\_Base\\_Start\\_IT\(\)](#)
- [HAL\\_TIM\\_Base\\_Stop\\_IT\(\)](#)
- [HAL\\_TIM\\_Base\\_Start\\_DMA\(\)](#)
- [HAL\\_TIM\\_Base\\_Stop\\_DMA\(\)](#)

### 39.2.4 Time Output Compare functions

This section provides functions allowing to:



- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the Time Output Compare.
- Stop the Time Output Compare.
- Start the Time Output Compare and enable interrupt.
- Stop the Time Output Compare and disable interrupt.
- Start the Time Output Compare and enable DMA transfer.
- Stop the Time Output Compare and disable DMA transfer.
- [\*HAL\\_TIM\\_OC\\_Init\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_DeInit\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_MspDeInit\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Stop\\_DMA\(\)\*](#)

### 39.2.5 Time PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM PWM.
- De-initialize the TIM PWM.
- Start the Time PWM.
- Stop the Time PWM.
- Start the Time PWM and enable interrupt.
- Stop the Time PWM and disable interrupt.
- Start the Time PWM and enable DMA transfer.
- Stop the Time PWM and disable DMA transfer.
- [\*HAL\\_TIM\\_PWM\\_Init\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_DeInit\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_MspDeInit\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_Stop\\_DMA\(\)\*](#)

### 39.2.6 Time Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the Time Input Capture.
- Stop the Time Input Capture.
- Start the Time Input Capture and enable interrupt.
- Stop the Time Input Capture and disable interrupt.



- Start the Time Input Capture and enable DMA transfer.
- Stop the Time Input Capture and disable DMA transfer.
- [\*HAL\\_TIM\\_IC\\_Init\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_DeInit\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_MspDeInit\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Stop\\_DMA\(\)\*](#)

### 39.2.7 Time One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the Time One Pulse.
- Stop the Time One Pulse.
- Start the Time One Pulse and enable interrupt.
- Stop the Time One Pulse and disable interrupt.
- Start the Time One Pulse and enable DMA transfer.
- Stop the Time One Pulse and disable DMA transfer.
- [\*HAL\\_TIM\\_OnePulse\\_Init\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_DeInit\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_MspDeInit\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Stop\\_IT\(\)\*](#)

### 39.2.8 Time Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the Time Encoder.
- Stop the Time Encoder.
- Start the Time Encoder and enable interrupt.
- Stop the Time Encoder and disable interrupt.
- Start the Time Encoder and enable DMA transfer.
- Stop the Time Encoder and disable DMA transfer.
- [\*HAL\\_TIM\\_Encoder\\_Init\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_DeInit\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_MspDeInit\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_Stop\(\)\*](#)

- [HAL\\_TIM\\_Encoder\\_Start\\_IT\(\)](#)
- [HAL\\_TIM\\_Encoder\\_Stop\\_IT\(\)](#)
- [HAL\\_TIM\\_Encoder\\_Start\\_DMA\(\)](#)
- [HAL\\_TIM\\_Encoder\\_Stop\\_DMA\(\)](#)

### 39.2.9 IRQ handler management

This section provides Timer IRQ handler function.

- [HAL\\_TIM\\_IRQHandler\(\)](#)

### 39.2.10 Peripheral Control functions

This section provides functions allowing to:

- Configure The Input Output channels for OC, PWM, IC or One Pulse mode.
- Configure External Clock source.
- Configure Complementary channels, break features and dead time.
- Configure Master and the Slave synchronization.
- Configure the DMA Burst Mode.
- [HAL\\_TIM\\_OC\\_ConfigChannel\(\)](#)
- [HAL\\_TIM\\_IC\\_ConfigChannel\(\)](#)
- [HAL\\_TIM\\_PWM\\_ConfigChannel\(\)](#)
- [HAL\\_TIM\\_OnePulse\\_ConfigChannel\(\)](#)
- [HAL\\_TIM\\_DMABurst\\_WriteStart\(\)](#)
- [HAL\\_TIM\\_DMABurst\\_WriteStop\(\)](#)
- [HAL\\_TIM\\_DMABurst\\_ReadStart\(\)](#)
- [HAL\\_TIM\\_DMABurst\\_ReadStop\(\)](#)
- [HAL\\_TIM\\_GenerateEvent\(\)](#)
- [HAL\\_TIM\\_ConfigOCrefClear\(\)](#)
- [HAL\\_TIM\\_ConfigClockSource\(\)](#)
- [HAL\\_TIM\\_ConfigTI1Input\(\)](#)
- [HAL\\_TIM\\_SlaveConfigSynchronization\(\)](#)
- [HAL\\_TIM\\_SlaveConfigSynchronization\\_IT\(\)](#)
- [HAL\\_TIM\\_ReadCapturedValue\(\)](#)

### 39.2.11 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\(\)](#)

## 39.2.12 Peripheral State functions

This subsection permit 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\(\)](#)

### 39.2.13 HAL\_TIM\_Base\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : TIM Base handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.14 HAL\_TIM\_Base\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes the TIM Base peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : TIM Base handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.15 HAL\_TIM\_Base\_MspInit

Function Name	<b>void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Base MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 39.2.16 HAL\_TIM\_Base\_MspDeInit

Function Name	<b>void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM Base MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : TIM handle</li> </ul>

Return values • None

### 39.2.17 HAL\_TIM\_Base\_Start

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Base\_Start (TIM\_HandleTypeDef \* htim)**

Function Description Starts the TIM Base generation.

Parameters • **htim**: : TIM handle

Return values • HAL status

### 39.2.18 HAL\_TIM\_Base\_Stop

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Base\_Stop (TIM\_HandleTypeDef \* htim)**

Function Description Stops the TIM Base generation.

Parameters • **htim**: : TIM handle

Return values • HAL status

### 39.2.19 HAL\_TIM\_Base\_Start\_IT

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Base\_Start\_IT (TIM\_HandleTypeDef \* htim)**

Function Description Starts the TIM Base generation in interrupt mode.

Parameters • **htim**: : TIM handle

Return values • HAL status

### 39.2.20 HAL\_TIM\_Base\_Stop\_IT

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Base\_Stop\_IT (TIM\_HandleTypeDef \* htim)**

Function Description Stops the TIM Base generation in interrupt mode.

Parameters • **htim**: : TIM handle

Return values • HAL status

### 39.2.21 HAL\_TIM\_Base\_Start\_DMA

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Base\_Start\_DMA (TIM\_HandleTypeDef \* htim, uint32\_t \* pData, uint16\_t Length)**

Function Description Starts the TIM Base generation in DMA mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> <li>• <b>pData</b>: : The source Buffer address.</li> <li>• <b>Length</b>: : The length of data to be transferred from memory to peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.22 HAL\_TIM\_Base\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)</b>
Function Description	Stops the TIM Base generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.23 HAL\_TIM\_OC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Output Compare handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.24 HAL\_TIM\_OC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Output Compare handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.25 HAL\_TIM\_OC\_MspInit

Function Name	<b>void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 39.2.26 HAL\_TIM\_OC\_MspDeInit

Function Name	<b>void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>htim:</b> : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 39.2.27 HAL\_TIM\_OC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"><li>• <b>htim:</b> : TIM Output Compare handle</li><li>• <b>Channel:</b> : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 39.2.28 HAL\_TIM\_OC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"><li>• <b>htim:</b> : TIM handle</li><li>• <b>Channel:</b> : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 39.2.29 HAL\_TIM\_OC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Output Compare signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htim:</b> : TIM OC handle</li><li>• <b>Channel:</b> : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected</li></ul>

TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

### 39.2.30 HAL\_TIM\_OC\_Stop\_IT

**Function Name** **HAL\_StatusTypeDef HAL\_TIM\_OC\_Stop\_IT**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)

**Function Description** Stops the TIM Output Compare signal generation in interrupt mode.

**Parameters**

- **htim**: : TIM Output Compare handle
- **Channel**: : TIM Channel to be disabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected TIM\_CHANNEL\_3: TIM Channel 3 selected TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

### 39.2.31 HAL\_TIM\_OC\_Start\_DMA

**Function Name** **HAL\_StatusTypeDef HAL\_TIM\_OC\_Start\_DMA**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel, uint32\_t \* pData, uint16\_t Length)

**Function Description** Starts the TIM Output Compare signal generation in DMA mode.

**Parameters**

- **htim**: : TIM Output Compare handle
- **Channel**: : TIM Channel to be enabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected TIM\_CHANNEL\_3: TIM Channel 3 selected TIM\_CHANNEL\_4: TIM Channel 4 selected
- **pData**: : The source Buffer address.
- **Length**: : The length of data to be transferred from memory to TIM peripheral

Return values

- HAL status

### 39.2.32 HAL\_TIM\_OC\_Stop\_DMA

**Function Name** **HAL\_StatusTypeDef HAL\_TIM\_OC\_Stop\_DMA**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)

**Function Description** Stops the TIM Output Compare signal generation in DMA mode.

**Parameters**

- **htim**: : TIM Output Compare handle
- **Channel**: : TIM Channel to be disabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected TIM\_CHANNEL\_3: TIM Channel 3 selected

---

TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

### 39.2.33 HAL\_TIM\_PWM\_Init

Function Name **HAL\_StatusTypeDef HAL\_TIM\_PWM\_Init (TIM\_HandleTypeDef \* htim)**

Function Description Initializes the TIM PWM Time Base according to the specified parameters in the TIM\_HandleTypeDef and create the associated handle.

Parameters

- **htim**: : TIM handle

Return values

- HAL status

### 39.2.34 HAL\_TIM\_PWM\_DeInit

Function Name **HAL\_StatusTypeDef HAL\_TIM\_PWM\_DeInit (TIM\_HandleTypeDef \* htim)**

Function Description DeInitializes the TIM peripheral.

Parameters

- **htim**: : TIM handle

Return values

- HAL status

### 39.2.35 HAL\_TIM\_PWM\_MspInit

Function Name **void HAL\_TIM\_PWM\_MspInit (TIM\_HandleTypeDef \* htim)**

Function Description Initializes the TIM PWM MSP.

Parameters

- **htim**: : TIM handle

Return values

- None

### 39.2.36 HAL\_TIM\_PWM\_MspDeInit

Function Name **void HAL\_TIM\_PWM\_MspDeInit (TIM\_HandleTypeDef \* htim)**

Function Description DeInitializes TIM PWM MSP.

Parameters

- **htim**: : TIM handle

Return values

- None

### 39.2.37 HAL\_TIM\_PWM\_Start

Function Name **HAL\_StatusTypeDef HAL\_TIM\_PWM\_Start (TIM\_HandleTypeDef \* htim, uint32\_t Channel)**



Function Description	Starts the PWM signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.38 HAL\_TIM\_PWM\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the PWM signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.39 HAL\_TIM\_PWM\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.40 HAL\_TIM\_PWM\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected</li> </ul>

TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

### 39.2.41 HAL\_TIM\_PWM\_Start\_DMA

Function Name

**HAL\_StatusTypeDef HAL\_TIM\_PWM\_Start\_DMA**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel, uint32\_t \* pData, uint16\_t Length)

Function Description

Starts the TIM PWM signal generation in DMA mode.

Parameters

- **htim**: : TIM handle
- **Channel**: : TIM Channels to be enabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected TIM\_CHANNEL\_3: TIM Channel 3 selected TIM\_CHANNEL\_4: TIM Channel 4 selected
- **pData**: : The source Buffer address.
- **Length**: : The length of data to be transferred from memory to TIM peripheral

Return values

- HAL status

### 39.2.42 HAL\_TIM\_PWM\_Stop\_DMA

Function Name

**HAL\_StatusTypeDef HAL\_TIM\_PWM\_Stop\_DMA**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)

Function Description

Stops the TIM PWM signal generation in DMA mode.

Parameters

- **htim**: : TIM handle
- **Channel**: : TIM Channels to be disabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected TIM\_CHANNEL\_3: TIM Channel 3 selected TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

### 39.2.43 HAL\_TIM\_IC\_Init

Function Name

**HAL\_StatusTypeDef HAL\_TIM\_IC\_Init** (TIM\_HandleTypeDef \* htim)

Function Description

Initializes the TIM Input Capture Time base according to the specified parameters in the TIM\_HandleTypeDef and create the associated handle.

Parameters

- **htim**: : TIM Input Capture handle

Return values

- HAL status

**39.2.44 HAL\_TIM\_IC\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM Input Capture handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**39.2.45 HAL\_TIM\_IC\_MspInit**

Function Name	<b>void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**39.2.46 HAL\_TIM\_IC\_MspDeInit**

Function Name	<b>void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**39.2.47 HAL\_TIM\_IC\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> <li><b>htim:</b> : TIM Input Capture handle</li> <li><b>Channel:</b> : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**39.2.48 HAL\_TIM\_IC\_Stop**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Input Capture measurement.

Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.49 HAL\_TIM\_IC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM Input Capture handle</li> <li>• <b>Channel:</b> : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.50 HAL\_TIM\_IC\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.51 HAL\_TIM\_IC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</b>
Function Description	Starts the TIM Input Capture measurement in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM Input Capture handle</li> <li>• <b>Channel:</b> : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected</li> </ul>

- TIM\_CHANNEL\_4: TIM Channel 4 selected
  - **pData**: : The destination Buffer address.
  - **Length**: : The length of data to be transferred from TIM peripheral to memory.
- Return values
- HAL status

### 39.2.52 HAL\_TIM\_IC\_Stop\_DMA

- Function Name **HAL\_StatusTypeDef HAL\_TIM\_IC\_Stop\_DMA (TIM\_HandleTypeDef \* htim, uint32\_t Channel)**
- Function Description Stops the TIM Input Capture measurement in DMA mode.
- Parameters
- **htim**: : TIM Input Capture handle
  - **Channel**: : TIM Channels to be disabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected TIM\_CHANNEL\_3: TIM Channel 3 selected TIM\_CHANNEL\_4: TIM Channel 4 selected
- Return values
- HAL status

### 39.2.53 HAL\_TIM\_OnePulse\_Init

- Function Name **HAL\_StatusTypeDef HAL\_TIM\_OnePulse\_Init (TIM\_HandleTypeDef \* htim, uint32\_t OnePulseMode)**
- Function Description Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM\_HandleTypeDef and create the associated handle.
- Parameters
- **htim**: : TIM OnePulse handle
  - **OnePulseMode**: : Select the One pulse mode. This parameter can be one of the following values:  
TIM\_OPMODE\_SINGLE: Only one pulse will be generated.  
TIM\_OPMODE\_REPETITIVE: Repetitive pulses will be generated.
- Return values
- HAL status

### 39.2.54 HAL\_TIM\_OnePulse\_DeInit

- Function Name **HAL\_StatusTypeDef HAL\_TIM\_OnePulse\_DeInit (TIM\_HandleTypeDef \* htim)**
- Function Description DeInitializes the TIM One Pulse.
- Parameters
- **htim**: : TIM One Pulse handle
- Return values
- HAL status

### 39.2.55 HAL\_TIM\_OnePulse\_MspInit

---

Function Name	<b>void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b>: : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 39.2.56 HAL\_TIM\_OnePulse\_MspDeInit

Function Name	<b>void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b>: : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 39.2.57 HAL\_TIM\_OnePulse\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
Function Description	Starts the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b>: : TIM One Pulse handle</li><li>• <b>OutputChannel</b>: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 39.2.58 HAL\_TIM\_OnePulse\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
Function Description	Stops the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b>: : TIM One Pulse handle</li><li>• <b>OutputChannel</b>: : TIM Channels to be disable This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 39.2.59 HAL\_TIM\_OnePulse\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
---------------	---

Function Description	Starts the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM One Pulse handle</li> <li>• <b>OutputChannel</b>: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.60 HAL\_TIM\_OnePulse\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
Function Description	Stops the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM One Pulse handle</li> <li>• <b>OutputChannel</b>: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.61 HAL\_TIM\_Encoder\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)</b>
Function Description	Initializes the TIM Encoder Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Encoder Interface handle</li> <li>• <b>sConfig</b>: : TIM Encoder Interface configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.62 HAL\_TIM\_Encoder\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes the TIM Encoder interface.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Encoder handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.63 HAL\_TIM\_Encoder\_MspInit

Function Name	<b>void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)</b>
---------------	--

Function Description	Initializes the TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 39.2.64 HAL\_TIM\_Encoder\_MspDeInit

Function Name	<b>void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 39.2.65 HAL\_TIM\_Encoder\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Encoder Interface.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Encoder Interface handle</li> <li>• <b>Channel</b>: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.66 HAL\_TIM\_Encoder\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Encoder Interface.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Encoder Interface handle</li> <li>• <b>Channel</b>: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.67 HAL\_TIM\_Encoder\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
---------------	--



Function Description	Starts the TIM Encoder Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM Encoder Interface handle</li> <li>• <b>Channel:</b> : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.68 HAL\_TIM\_Encoder\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Encoder Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM Encoder Interface handle</li> <li>• <b>Channel:</b> : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.69 HAL\_TIM\_Encoder\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)</b>
Function Description	Starts the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM Encoder Interface handle</li> <li>• <b>Channel:</b> : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected</li> <li>• <b>pData1:</b> : The destination Buffer address for IC1.</li> <li>• <b>pData2:</b> : The destination Buffer address for IC2.</li> <li>• <b>Length:</b> : The length of data to be transferred from TIM peripheral to memory.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.70 HAL\_TIM\_Encoder\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Encoder Interface in DMA mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Encoder Interface handle</li> <li>• <b>Channel</b>: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.71 HAL\_TIM\_IRQHandler

Function Name	<b>void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)</b>
Function Description	This function handles TIM interrupts requests.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 39.2.72 HAL\_TIM\_OC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)</b>
Function Description	Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Output Compare handle</li> <li>• <b>sConfig</b>: : TIM Output Compare configuration structure</li> <li>• <b>Channel</b>: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.73 HAL\_TIM\_IC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig, uint32_t Channel)</b>
Function Description	Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM IC handle</li> <li>• <b>sConfig</b>: : TIM Input Capture configuration structure</li> <li>• <b>Channel</b>: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>

Return values

- HAL status

### 39.2.74 HAL\_TIM\_PWM\_ConfigChannel

**Function Name** `HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel(TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)`

**Function Description** Initializes the TIM PWM channels according to the specified parameters in the TIM\_OC\_InitTypeDef.

**Parameters**

- **htim**: : TIM handle
- **sConfig**: : TIM PWM configuration structure
- **Channel**: : TIM Channels to be enabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected TIM\_CHANNEL\_3: TIM Channel 3 selected TIM\_CHANNEL\_4: TIM Channel 4 selected

**Return values**

- HAL status

### 39.2.75 HAL\_TIM\_OnePulse\_ConfigChannel

**Function Name** `HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel(TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)`

**Function Description** Initializes the TIM One Pulse Channels according to the specified parameters in the TIM\_OnePulse\_InitTypeDef.

**Parameters**

- **htim**: : TIM One Pulse handle
- **sConfig**: : TIM One Pulse configuration structure
- **OutputChannel**: : TIM Channels to be enabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected
- **InputChannel**: : TIM Channels to be enabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected

**Return values**

- HAL status

### 39.2.76 HAL\_TIM\_DMABurst\_WriteStart

**Function Name** `HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart(TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)`

**Function Description** Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

## Parameters

- **htim** : TIM handle
- **BurstBaseAddress** : TIM Base address from where the DMA will start the Data write This parameter can be one of the following values: TIM\_DMABASE\_CR1  
TIM\_DMABASE\_CR2 TIM\_DMABASE\_SMCR  
TIM\_DMABASE\_DIER TIM\_DMABASE\_SR  
TIM\_DMABASE\_EGR TIM\_DMABASE\_CCMR1  
TIM\_DMABASE\_CCMR2 TIM\_DMABASE\_CCER  
TIM\_DMABASE\_CNT TIM\_DMABASE\_PSC  
TIM\_DMABASE\_ARR TIM\_DMABASE\_RCR  
TIM\_DMABASE\_CCR1 TIM\_DMABASE\_CCR2  
TIM\_DMABASE\_CCR3 TIM\_DMABASE\_CCR4  
TIM\_DMABASE\_BDTR TIM\_DMABASE\_DCR
- **BurstRequestSrc** : TIM DMA Request sources This parameter can be one of the following values:  
TIM\_DMA\_UPDATE: TIM update Interrupt source  
TIM\_DMA\_CC1: TIM Capture Compare 1 DMA source  
TIM\_DMA\_CC2: TIM Capture Compare 2 DMA source  
TIM\_DMA\_CC3: TIM Capture Compare 3 DMA source  
TIM\_DMA\_CC4: TIM Capture Compare 4 DMA source  
TIM\_DMA\_COM: TIM Commutation DMA source  
TIM\_DMA\_TRIGGER: TIM Trigger DMA source
- **BurstBuffer** : The Buffer address.
- **BurstLength** : DMA Burst length. This parameter can be one value between: TIM\_DMABURSTLENGTH\_1TRANSFER and TIM\_DMABURSTLENGTH\_18TRANSFERS.

## Return values

- HAL status

### 39.2.77 HAL\_TIM\_DMABurst\_WriteStop

## Function Name

**HAL\_StatusTypeDef HAL\_TIM\_DMABurst\_WriteStop**  
(TIM\_HandleTypeDef \* htim, uint32\_t BurstRequestSrc)

## Function Description

Stops the TIM DMA Burst mode.

## Parameters

- **htim** : TIM handle
- **BurstRequestSrc** : TIM DMA Request sources to disable

## Return values

- HAL status

### 39.2.78 HAL\_TIM\_DMABurst\_ReadStart

## Function Name

**HAL\_StatusTypeDef HAL\_TIM\_DMABurst\_ReadStart**  
(TIM\_HandleTypeDef \* htim, uint32\_t BurstBaseAddress,  
uint32\_t BurstRequestSrc, uint32\_t \* BurstBuffer, uint32\_t  
BurstLength)

## Function Description

Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

## Parameters

- **htim** : TIM handle
- **BurstBaseAddress** : TIM Base address from where the DMA will starts the Data read This parameter can be one of the following values: TIM\_DMABASE\_CR1

TIM\_DMABASE\_CR2 TIM\_DMABASE\_SMCR  
TIM\_DMABASE\_DIER TIM\_DMABASE\_SR  
TIM\_DMABASE\_EGR TIM\_DMABASE\_CCMR1  
TIM\_DMABASE\_CCMR2 TIM\_DMABASE\_CCER  
TIM\_DMABASE\_CNT TIM\_DMABASE\_PSC  
TIM\_DMABASE\_ARR TIM\_DMABASE\_RCR  
TIM\_DMABASE\_CCR1 TIM\_DMABASE\_CCR2  
TIM\_DMABASE\_CCR3 TIM\_DMABASE\_CCR4  
TIM\_DMABASE\_BDTR TIM\_DMABASE\_DCR

- **BurstRequestSrc**: : TIM DMA Request sources This parameter can be one of the following values:  
TIM\_DMA\_UPDATE: TIM update Interrupt source  
TIM\_DMA\_CC1: TIM Capture Compare 1 DMA source  
TIM\_DMA\_CC2: TIM Capture Compare 2 DMA source  
TIM\_DMA\_CC3: TIM Capture Compare 3 DMA source  
TIM\_DMA\_CC4: TIM Capture Compare 4 DMA source  
TIM\_DMA\_COM: TIM Commutation DMA source  
TIM\_DMA\_TRIGGER: TIM Trigger DMA source
- **BurstBuffer**: : The Buffer address.
- **BurstLength**: : DMA Burst length. This parameter can be one value between: TIM\_DMABURSTLENGTH\_1TRANSFER and TIM\_DMABURSTLENGTH\_18TRANSFERS.

Return values

- HAL status

### 39.2.79 HAL\_TIM\_DMABurst\_ReadStop

**Function Name** HAL\_StatusTypeDef HAL\_TIM\_DMABurst\_ReadStop  
(TIM\_HandleTypeDef \* htim, uint32\_t BurstRequestSrc)

**Function Description** Stop the DMA burst reading.

**Parameters**

- **htim**: : TIM handle
- **BurstRequestSrc**: : TIM DMA Request sources to disable.

**Return values**

- HAL status

### 39.2.80 HAL\_TIM\_GenerateEvent

**Function Name** HAL\_StatusTypeDef HAL\_TIM\_GenerateEvent  
(TIM\_HandleTypeDef \* htim, uint32\_t EventSource)

**Function Description** Generate a software event.

**Parameters**

- **htim**: : TIM handle
- **EventSource**: : specifies the event source. This parameter can be one of the following values:  
TIM\_EVENTSOURCE\_UPDATE: Timer update Event source  
TIM\_EVENTSOURCE\_CC1: Timer Capture Compare 1 Event source  
TIM\_EVENTSOURCE\_CC2: Timer Capture Compare 2 Event source  
TIM\_EVENTSOURCE\_CC3: Timer Capture Compare 3 Event source  
TIM\_EVENTSOURCE\_CC4: Timer Capture Compare 4 Event source  
TIM\_EVENTSOURCE\_COM: Timer COM event source  
TIM\_EVENTSOURCE\_TRIGGER: Timer Trigger Event

source TIM\_EVENTSOURCE\_BREAK: Timer Break event source

## Return values

- HAL status

## Notes

- TIM6 and TIM7 can only generate an update event.
- TIM\_EVENTSOURCE\_COM and TIM\_EVENTSOURCE\_BREAK are used only with TIM1, TIM15, TIM16 and TIM17.

### 39.2.81 HAL\_TIM\_ConfigOCrefClear

## Function Name

**HAL\_StatusTypeDef HAL\_TIM\_ConfigOCrefClear**  
(TIM\_HandleTypeDef \* htim, TIM\_ClearInputConfigTypeDef \* sClearInputConfig, uint32\_t Channel)

## Function Description

Configures the OCRef clear feature.

## Parameters

- **htim**: : TIM handle
- **sClearInputConfig**: : pointer to a TIM\_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel**: : specifies the TIM Channel This parameter can be one of the following values: TIM\_Channel\_1: TIM Channel 1 TIM\_Channel\_2: TIM Channel 2 TIM\_Channel\_3: TIM Channel 3 TIM\_Channel\_4: TIM Channel 4

## Return values

- HAL status

### 39.2.82 HAL\_TIM\_ConfigClockSource

## Function Name

**HAL\_StatusTypeDef HAL\_TIM\_ConfigClockSource**  
(TIM\_HandleTypeDef \* htim, TIM\_ClockConfigTypeDef \* sClockSourceConfig)

## Function Description

Configures the clock source to be used.

## Parameters

- **htim**: : TIM handle
- **sClockSourceConfig**: : pointer to a TIM\_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.

## Return values

- HAL status

### 39.2.83 HAL\_TIM\_ConfigTI1Input

## Function Name

**HAL\_StatusTypeDef HAL\_TIM\_ConfigTI1Input**  
(TIM\_HandleTypeDef \* htim, uint32\_t TI1\_Selection)

## Function Description

Selects the signal connected to the TI1 input: direct from CH1\_input or a XOR combination between CH1\_input, CH2\_input & CH3\_input.

## Parameters

- **htim**: : TIM handle.
- **TI1\_Selection**: : Indicate whether or not channel 1 is

connected to the output of a XOR gate. This parameter can be one of the following values: TIM\_TI1SELECTION\_CH1: The TIMx\_CH1 pin is connected to TI1 input  
TIM\_TI1SELECTION\_XORCOMBINATION: The TIMx\_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

Return values

- HAL status

### 39.2.84 HAL\_TIM\_SlaveConfigSynchronization

**Function Name** `HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization(TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)`

**Function Description** Configures the TIM in Slave mode.

**Parameters**

- **htim**: : TIM handle.
- **sSlaveConfig**: : pointer to a TIM\_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the ) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values

- HAL status

### 39.2.85 HAL\_TIM\_SlaveConfigSynchronization\_IT

**Function Name** `HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT(TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)`

**Function Description** Configures the TIM in Slave mode in interrupt mode.

**Parameters**

- **htim**: TIM handle.
- **sSlaveConfig**: pointer to a TIM\_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the ) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values

- HAL status

### 39.2.86 HAL\_TIM\_ReadCapturedValue

**Function Name** `uint32_t HAL_TIM_ReadCapturedValue(TIM_HandleTypeDef * htim, uint32_t Channel)`

**Function Description** Read the captured value from Capture Compare unit.

**Parameters**

- **htim**: : TIM handle.
- **Channel**: : TIM Channels to be enabled This parameter can be one of the following values: TIM\_CHANNEL\_1 : TIM

Channel 1 selected TIM\_CHANNEL\_2 : TIM Channel 2  
 selected TIM\_CHANNEL\_3 : TIM Channel 3 selected  
 TIM\_CHANNEL\_4 : TIM Channel 4 selected

Return values

- Captured value

### 39.2.87 HAL\_TIM\_PeriodElapsedCallback

Function Name **void HAL\_TIM\_PeriodElapsedCallback (TIM\_HandleTypeDef \* htim)**

Function Description Period elapsed callback in non blocking mode.

Parameters

- **htim**: : TIM handle

Return values

- None

### 39.2.88 HAL\_TIM\_OC\_DelayElapsedCallback

Function Name **void HAL\_TIM\_OC\_DelayElapsedCallback (TIM\_HandleTypeDef \* htim)**

Function Description Output Compare callback in non blocking mode.

Parameters

- **htim**: : TIM OC handle

Return values

- None

### 39.2.89 HAL\_TIM\_IC\_CaptureCallback

Function Name **void HAL\_TIM\_IC\_CaptureCallback (TIM\_HandleTypeDef \* htim)**

Function Description Input Capture callback in non blocking mode.

Parameters

- **htim**: : TIM IC handle

Return values

- None

### 39.2.90 HAL\_TIM\_PWM\_PulseFinishedCallback

Function Name **void HAL\_TIM\_PWM\_PulseFinishedCallback (TIM\_HandleTypeDef \* htim)**

Function Description PWM Pulse finished callback in non blocking mode.

Parameters

- **htim**: : TIM handle

Return values

- None

### 39.2.91 HAL\_TIM\_TriggerCallback

Function Name **void HAL\_TIM\_TriggerCallback (TIM\_HandleTypeDef \* htim)**



Function Description	Hall Trigger detection callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 39.2.92 HAL\_TIM\_ErrorCallback

Function Name	<b>void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)</b>
Function Description	Timer error callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 39.2.93 HAL\_TIM\_Base\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)</b>
Function Description	Return the TIM Base state.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Base handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

### 39.2.94 HAL\_TIM\_OC\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)</b>
Function Description	Return the TIM OC state.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Output Compare handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

### 39.2.95 HAL\_TIM\_PWM\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)</b>
Function Description	Return the TIM PWM state.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

### 39.2.96 HAL\_TIM\_IC\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)</b>
---------------	--

Function Description	Return the TIM Input Capture state.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM IC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

### 39.2.97 HAL\_TIM\_OnePulse\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)</b>
Function Description	Return the TIM One Pulse Mode state.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM OPM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

### 39.2.98 HAL\_TIM\_Encoder\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)</b>
Function Description	Return the TIM Encoder Mode state.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Encoder handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

## 39.3 TIM Firmware driver defines

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

### 39.3.1 TIM

TIM

***TIM Automatic Output Enable***

TIM\_AUTOMATICOUTPUT\_ENABLE

TIM\_AUTOMATICOUTPUT\_DISABLE

***TIM Break Input Enable Disable***

TIM\_BREAK\_ENABLE

TIM\_BREAK\_DISABLE

***TIM Break Input Polarity***

TIM\_BREAKPOLARITY\_LOW

TIM\_BREAKPOLARITY\_HIGH

***TIM Channel***

TIM\_CHANNEL\_1

TIM\_CHANNEL\_2

TIM\_CHANNEL\_3

TIM\_CHANNEL\_4

TIM\_CHANNEL\_ALL

**TIM Capture/Compare Channel State**

TIM\_CCx\_ENABLE

TIM\_CCx\_DISABLE

TIM\_CCxN\_ENABLE

TIM\_CCxN\_DISABLE

**TIM Clear Input Polarity**

TIM\_CLEARINPUTPOLARITY\_INVERTED      Polarity for ETRx pin

TIM\_CLEARINPUTPOLARITY\_NONINVERTED      Polarity for ETRx pin

**TIM Clear Input Prescaler**

TIM\_CLEARINPUTPRESCALER\_DIV1      No prescaler is used

TIM\_CLEARINPUTPRESCALER\_DIV2      Prescaler for External ETR pin: Capture performed once every 2 events.

TIM\_CLEARINPUTPRESCALER\_DIV4      Prescaler for External ETR pin: Capture performed once every 4 events.

TIM\_CLEARINPUTPRESCALER\_DIV8      Prescaler for External ETR pin: Capture performed once every 8 events.

**TIM ClearInput Source**

TIM\_CLEARINPUTSOURCE\_ETR

TIM\_CLEARINPUTSOURCE\_OCREFCLR

TIM\_CLEARINPUTSOURCE\_NONE

**TIM ClockDivision**

TIM\_CLOCKDIVISION\_DIV1

TIM\_CLOCKDIVISION\_DIV2

TIM\_CLOCKDIVISION\_DIV4

**TIM Clock Polarity**

TIM\_CLOCKPOLARITY\_INVERTED      Polarity for ETRx clock sources

TIM\_CLOCKPOLARITY\_NONINVERTED      Polarity for ETRx clock sources

TIM\_CLOCKPOLARITY\_RISING      Polarity for Tlx clock sources

TIM\_CLOCKPOLARITY\_FALLING      Polarity for Tlx clock sources

TIM\_CLOCKPOLARITY\_BOTHEDGE      Polarity for Tlx clock sources

**TIM Clock Prescaler**

TIM\_CLOCKPRESCALER\_DIV1      No prescaler is used

TIM\_CLOCKPRESCALER\_DIV2      Prescaler for External ETR Clock: Capture performed once every 2 events.

TIM\_CLOCKPRESCALER\_DIV4      Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM\_CLOCKPRESCALER\_DIV8 Prescaler for External ETR Clock: Capture performed once every 8 events.

***TIM Clock Source***

TIM\_CLOCKSOURCE\_ETRMODE2

TIM\_CLOCKSOURCE\_INTERNAL

TIM\_CLOCKSOURCE\_ITR0

TIM\_CLOCKSOURCE\_ITR1

TIM\_CLOCKSOURCE\_ITR2

TIM\_CLOCKSOURCE\_ITR3

TIM\_CLOCKSOURCE\_TI1ED

TIM\_CLOCKSOURCE\_TI1

TIM\_CLOCKSOURCE\_TI2

TIM\_CLOCKSOURCE\_ETRMODE1

***TIM Commutation Source***

TIM\_COMMUTATION\_TRGI

TIM\_COMMUTATION\_SOFTWARE

***TIM Counter Mode***

TIM\_COUNTERMODE\_UP

TIM\_COUNTERMODE\_DOWN

TIM\_COUNTERMODE\_CENTERALIGNED1

TIM\_COUNTERMODE\_CENTERALIGNED2

TIM\_COUNTERMODE\_CENTERALIGNED3

***TIM DMA Base Address***

TIM\_DMABASE\_CR1

TIM\_DMABASE\_CR2

TIM\_DMABASE\_SMCR

TIM\_DMABASE\_DIER

TIM\_DMABASE\_SR

TIM\_DMABASE\_EGR

TIM\_DMABASE\_CCMR1

TIM\_DMABASE\_CCMR2

TIM\_DMABASE\_CCER

TIM\_DMABASE\_CNT

TIM\_DMABASE\_PSC

TIM\_DMABASE\_ARR

TIM\_DMABASE\_RCR

TIM\_DMABASE\_CCR1

TIM\_DMABASE\_CCR2

TIM\_DMABASE\_CCR3

TIM\_DMABASE\_CCR4

TIM\_DMABASE\_BDTR

TIM\_DMABASE\_DCR

**TIM DMA Burst Length**

TIM\_DMABURSTLENGTH\_1TRANSFER

TIM\_DMABURSTLENGTH\_2TRANSFERS

TIM\_DMABURSTLENGTH\_3TRANSFERS

TIM\_DMABURSTLENGTH\_4TRANSFERS

TIM\_DMABURSTLENGTH\_5TRANSFERS

TIM\_DMABURSTLENGTH\_6TRANSFERS

TIM\_DMABURSTLENGTH\_7TRANSFERS

TIM\_DMABURSTLENGTH\_8TRANSFERS

TIM\_DMABURSTLENGTH\_9TRANSFERS

TIM\_DMABURSTLENGTH\_10TRANSFERS

TIM\_DMABURSTLENGTH\_11TRANSFERS

TIM\_DMABURSTLENGTH\_12TRANSFERS

TIM\_DMABURSTLENGTH\_13TRANSFERS

TIM\_DMABURSTLENGTH\_14TRANSFERS

TIM\_DMABURSTLENGTH\_15TRANSFERS

TIM\_DMABURSTLENGTH\_16TRANSFERS

TIM\_DMABURSTLENGTH\_17TRANSFERS

TIM\_DMABURSTLENGTH\_18TRANSFERS

**TIM DMA Handle Index**

TIM\_DMA\_ID\_UPDATE      Index of the DMA handle used for Update DMA requests

TIM\_DMA\_ID\_CC1      Index of the DMA handle used for Capture/Compare 1  
DMA requestsTIM\_DMA\_ID\_CC2      Index of the DMA handle used for Capture/Compare 2  
DMA requestsTIM\_DMA\_ID\_CC3      Index of the DMA handle used for Capture/Compare 3  
DMA requestsTIM\_DMA\_ID\_CC4      Index of the DMA handle used for Capture/Compare 4  
DMA requestsTIM\_DMA\_ID\_COMMUTATION      Index of the DMA handle used for Commutation DMA  
requests

TIM\_DMA\_ID\_TRIGGER      Index of the DMA handle used for Trigger DMA requests

**TIM DMA Sources**

TIM\_DMA\_UPDATE

TIM\_DMA\_CC1

TIM\_DMA\_CC2

TIM\_DMA\_CC3

TIM\_DMA\_CC4

TIM\_DMA\_COM

TIM\_DMA\_TRIGGER

**TIM Encoder Mode**

TIM\_ENCODERMODE\_TI1

TIM\_ENCODERMODE\_TI2

TIM\_ENCODERMODE\_TI12

**TIM ETR Polarity**

TIM\_ETRPOLARITY\_INVERTED      Polarity for ETR source

TIM\_ETRPOLARITY\_NONINVERTED      Polarity for ETR source

**TIM ETR Prescaler**

TIM\_ETRPRESCALER\_DIV1      No prescaler is used

TIM\_ETRPRESCALER\_DIV2      ETR input source is divided by 2

TIM\_ETRPRESCALER\_DIV4      ETR input source is divided by 4

TIM\_ETRPRESCALER\_DIV8      ETR input source is divided by 8

**TIM Event Source**

TIM\_EVENTSOURCE\_UPDATE

TIM\_EVENTSOURCE\_CC1

TIM\_EVENTSOURCE\_CC2

TIM\_EVENTSOURCE\_CC3

TIM\_EVENTSOURCE\_CC4

TIM\_EVENTSOURCE\_COM

TIM\_EVENTSOURCE\_TRIGGER

TIM\_EVENTSOURCE\_BREAK

**TIM Exported Macros**

\_\_HAL\_TIM\_RESET\_HANDLE\_STATE

**Description:**

- Reset TIM handle state.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.

**Return value:**

- None:

\_\_HAL\_TIM\_ENABLE

**Description:**

- Enable the TIM peripheral.

**Parameters:**

- `__HANDLE__`: TIM handle

**Return value:**

- None:

**Description:**

- Enable the TIM main Output.

**Parameters:**

- `__HANDLE__`: TIM handle

**Return value:**

- None:

**Description:**

- Disable the TIM peripheral.

**Parameters:**

- `__HANDLE__`: TIM handle

**Return value:**

- None:

**Description:**

- Disable the TIM main Output.

**Parameters:**

- `__HANDLE__`: TIM handle

**Return value:**

- None:

**Description:**

- Enables the specified TIM interrupt.

**Parameters:**

- `__HANDLE__`: specifies the TIM Handle.
- `__INTERRUPT__`: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
  - `TIM_IT_UPDATE`: Update interrupt
  - `TIM_IT_CC1`: Capture/Compare 1 interrupt
  - `TIM_IT_CC2`: Capture/Compare 2 interrupt
  - `TIM_IT_CC3`: Capture/Compare 3 interrupt
  - `TIM_IT_CC4`: Capture/Compare 4 interrupt
  - `TIM_IT_COM`: Commutation interrupt
  - `TIM_IT_TRIGGER`: Trigger interrupt
  - `TIM_IT_BREAK`: Break interrupt

**Return value:**`__HAL_TIM_MOE_ENABLE``__HAL_TIM_DISABLE``__HAL_TIM_MOE_DISABLE``__HAL_TIM_ENABLE_IT`

**\_\_HAL\_TIM\_DISABLE\_IT**

- None:

**Description:**

- Disables the specified TIM interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the TIM Handle.
- **\_\_INTERRUPT\_\_**: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
  - TIM\_IT\_UPDATE: Update interrupt
  - TIM\_IT\_CC1: Capture/Compare 1 interrupt
  - TIM\_IT\_CC2: Capture/Compare 2 interrupt
  - TIM\_IT\_CC3: Capture/Compare 3 interrupt
  - TIM\_IT\_CC4: Capture/Compare 4 interrupt
  - TIM\_IT\_COM: Commutation interrupt
  - TIM\_IT\_TRIGGER: Trigger interrupt
  - TIM\_IT\_BREAK: Break interrupt

**Return value:**

- None:

**\_\_HAL\_TIM\_ENABLE\_DMA****Description:**

- Enables the specified DMA request.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the TIM Handle.
- **\_\_DMA\_\_**: specifies the TIM DMA request to enable. This parameter can be one of the following values:
  - TIM\_DMA\_UPDATE: Update DMA request
  - TIM\_DMA\_CC1: Capture/Compare 1 DMA request
  - TIM\_DMA\_CC2: Capture/Compare 2 DMA request
  - TIM\_DMA\_CC3: Capture/Compare 3 DMA request
  - TIM\_DMA\_CC4: Capture/Compare 4 DMA request
  - TIM\_DMA\_COM: Commutation DMA request
  - TIM\_DMA\_TRIGGER: Trigger DMA request

**Return value:**

- None:

**\_\_HAL\_TIM\_DISABLE\_DMA****Description:**

- Disables the specified DMA request.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the TIM Handle.
- **\_\_DMA\_\_**: specifies the TIM DMA request to disable. This parameter can be one of the



following values:

- TIM\_DMA\_UPDATE: Update DMA request
- TIM\_DMA\_CC1: Capture/Compare 1 DMA request
- TIM\_DMA\_CC2: Capture/Compare 2 DMA request
- TIM\_DMA\_CC3: Capture/Compare 3 DMA request
- TIM\_DMA\_CC4: Capture/Compare 4 DMA request
- TIM\_DMA\_COM: Commutation DMA request
- TIM\_DMA\_TRIGGER: Trigger DMA request

**Return value:**

- None:

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the TIM Handle.
- `__FLAG__`: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
  - TIM\_FLAG\_UPDATE: Update interrupt flag
  - TIM\_FLAG\_CC1: Capture/Compare 1 interrupt flag
  - TIM\_FLAG\_CC2: Capture/Compare 2 interrupt flag
  - TIM\_FLAG\_CC3: Capture/Compare 3 interrupt flag
  - TIM\_FLAG\_CC4: Capture/Compare 4 interrupt flag
  - TIM\_FLAG\_COM: Commutation interrupt flag
  - TIM\_FLAG\_TRIGGER: Trigger interrupt flag
  - TIM\_FLAG\_BREAK: Break interrupt flag
  - TIM\_FLAG\_CC1OF: Capture/Compare 1 overcapture flag
  - TIM\_FLAG\_CC2OF: Capture/Compare 2 overcapture flag
  - TIM\_FLAG\_CC3OF: Capture/Compare 3 overcapture flag
  - TIM\_FLAG\_CC4OF: Capture/Compare 4 overcapture flag

**Return value:**

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

**Description:**

`__HAL_TIM_GET_FLAG`

`__HAL_TIM_CLEAR_FLAG`

- Clears the specified TIM interrupt flag.

**Parameters:**

- `__HANDLE__`: specifies the TIM Handle.
- `__FLAG__`: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
  - `TIM_FLAG_UPDATE`: Update interrupt flag
  - `TIM_FLAG_CC1`: Capture/Compare 1 interrupt flag
  - `TIM_FLAG_CC2`: Capture/Compare 2 interrupt flag
  - `TIM_FLAG_CC3`: Capture/Compare 3 interrupt flag
  - `TIM_FLAG_CC4`: Capture/Compare 4 interrupt flag
  - `TIM_FLAG_COM`: Commutation interrupt flag
  - `TIM_FLAG_TRIGGER`: Trigger interrupt flag
  - `TIM_FLAG_BREAK`: Break interrupt flag
  - `TIM_FLAG_CC1OF`: Capture/Compare 1 overcapture flag
  - `TIM_FLAG_CC2OF`: Capture/Compare 2 overcapture flag
  - `TIM_FLAG_CC3OF`: Capture/Compare 3 overcapture flag
  - `TIM_FLAG_CC4OF`: Capture/Compare 4 overcapture flag

**Return value:**

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

`__HAL_TIM_GET_IT_SOURCE`**Description:**

- Checks whether the specified TIM interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: TIM handle
- `__INTERRUPT__`: specifies the TIM interrupt source to check.

**Return value:**

- The: state of `TIM_IT` (SET or RESET).

`__HAL_TIM_CLEAR_IT`**Description:**

- Clear the TIM interrupt pending bits.

**Parameters:**

- `__HANDLE__`: TIM handle
- `__INTERRUPT__`: specifies the interrupt pending bit to clear.

**Return value:**

`__HAL_TIM_IS_TIM_COUNTING_DOWN`

- None:

**Description:**

- Indicates whether or not the TIM Counter is used as downcounter.

**Parameters:**

- `__HANDLE__`: TIM handle.

**Return value:**

- False: (Counter used as upcounter) or True (Counter used as downcounter)

`__HAL_TIM_SET_PRESCALER`

**Description:**

- Sets the TIM active prescaler register value on update event.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__PRESC__`: specifies the active prescaler register new value.

**Return value:**

- None:

`__HAL_TIM_SET_COMPARE`

**Description:**

- Sets the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:
  - `TIM_CHANNEL_1`: TIM Channel 1 selected
  - `TIM_CHANNEL_2`: TIM Channel 2 selected
  - `TIM_CHANNEL_3`: TIM Channel 3 selected
  - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `__COMPARE__`: specifies the Capture Compare register new value.

**Return value:**

- None:

`__HAL_TIM_GET_COMPARE`

**Description:**

- Gets the TIM Capture Compare Register value on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channel associated with the capture compare register. This parameter can be one of the following values:
  - `TIM_CHANNEL_1`: get capture/compare 1 register value
  - `TIM_CHANNEL_2`: get capture/compare 2 register value
  - `TIM_CHANNEL_3`: get capture/compare 3 register value
  - `TIM_CHANNEL_4`: get capture/compare 4 register value

**Return value:**

- None:

**Description:**

- Sets the TIM Counter Register value on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__COUNTER__`: specifies the Counter register new value.

**Return value:**

- None:

**Description:**

- Gets the TIM Counter Register value on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.

**Return value:**

- None:

**Description:**

- Sets the TIM Autoreload Register value on runtime without calling another time any Init function.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__AUTORELOAD__`: specifies the Counter register new value.

**Return value:**

- None:

**Description:**

- Gets the TIM Autoreload Register value on

`__HAL_TIM_SET_COUNTER``__HAL_TIM_GET_COUNTER``__HAL_TIM_SET_AUTORELOAD``__HAL_TIM_GET_AUTORELOAD`

runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.

**Return value:**

- None:

`__HAL_TIM_SET_CLOCKDIVISION`  
N

**Description:**

- Sets the TIM Clock Division value on runtime without calling another time any Init function.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__CKD__`: specifies the clock division value. This parameter can be one of the following value:
  - `TIM_CLOCKDIVISION_DIV1`
  - `TIM_CLOCKDIVISION_DIV2`
  - `TIM_CLOCKDIVISION_DIV4`

**Return value:**

- None:

`__HAL_TIM_GET_CLOCKDIVISION`  
N

**Description:**

- Gets the TIM Clock Division value on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.

**Return value:**

- None:

`__HAL_TIM_SET_ICPRESCALER`

**Description:**

- Sets the TIM Input Capture prescaler on runtime without calling another time

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: : TIM Channels to be configured. This parameter can be one of the following values:
  - `TIM_CHANNEL_1`: TIM Channel 1 selected
  - `TIM_CHANNEL_2`: TIM Channel 2 selected
  - `TIM_CHANNEL_3`: TIM Channel 3 selected
  - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `__ICPSC__`: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
  - `TIM_ICPSC_DIV1`: no prescaler

- TIM\_ICPSC\_DIV2: capture is done once every 2 events
- TIM\_ICPSC\_DIV4: capture is done once every 4 events
- TIM\_ICPSC\_DIV8: capture is done once every 8 events

**Return value:**

- None:

**\_\_HAL\_TIM\_GET\_ICPRESCALER****Description:**

- Gets the TIM Input Capture prescaler on runtime.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.
- \_\_CHANNEL\_\_: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM\_CHANNEL\_1: get input capture 1 prescaler value
  - TIM\_CHANNEL\_2: get input capture 2 prescaler value
  - TIM\_CHANNEL\_3: get input capture 3 prescaler value
  - TIM\_CHANNEL\_4: get input capture 4 prescaler value

**Return value:**

- None:

**\_\_HAL\_TIM\_URS\_ENABLE****Description:**

- Set the Update Request Source (URS) bit of the TIMx\_CR1 register.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.

**Return value:**

- None:

**\_\_HAL\_TIM\_URS\_DISABLE****Description:**

- Reset the Update Request Source (URS) bit of the TIMx\_CR1 register.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.

**Return value:**

- None:

**\_\_HAL\_TIM\_SET\_CAPTUREPOLARITY****Description:**

- Sets the TIM Capture x input polarity on

runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:
  - `TIM_CHANNEL_1`: TIM Channel 1 selected
  - `TIM_CHANNEL_2`: TIM Channel 2 selected
  - `TIM_CHANNEL_3`: TIM Channel 3 selected
  - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `__POLARITY__`: Polarity for TIx source
  - `TIM_INPUTCHANNELPOLARITY_RISING`: Rising Edge
  - `TIM_INPUTCHANNELPOLARITY_FALLING`: Falling Edge
  - `TIM_INPUTCHANNELPOLARITY_BOTHEDGE`: Rising and Falling Edge

**Return value:**

- None:

***TIM Flag Definition***

`TIM_FLAG_UPDATE`

`TIM_FLAG_CC1`

`TIM_FLAG_CC2`

`TIM_FLAG_CC3`

`TIM_FLAG_CC4`

`TIM_FLAG_COM`

`TIM_FLAG_TRIGGER`

`TIM_FLAG_BREAK`

`TIM_FLAG_CC1OF`

`TIM_FLAG_CC2OF`

`TIM_FLAG_CC3OF`

`TIM_FLAG_CC4OF`

***TIM Input Capture Polarity***

`TIM_ICPOLARITY_RISING`

`TIM_ICPOLARITY_FALLING`

`TIM_ICPOLARITY_BOTHEDGE`

***TIM Input Capture Prescaler***

`TIM_ICPSC_DIV1` Capture performed each time an edge is detected on the capture

input

TIM_ICPSC_DIV2	Capture performed once every 2 events
TIM_ICPSC_DIV4	Capture performed once every 4 events
TIM_ICPSC_DIV8	Capture performed once every 8 events

**TIM Input Capture Selection**

TIM_ICSELECTION_DIRECTTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively
TIM_ICSELECTION_INDIRECTTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively
TIM_ICSELECTION_TRC	TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

**TIM Input Channel Polarity**

TIM_INPUTCHANNELPOLARITY_RISING	Polarity for Tlx source
TIM_INPUTCHANNELPOLARITY_FALLING	Polarity for Tlx source
TIM_INPUTCHANNELPOLARITY_BOTHEDGE	Polarity for Tlx source

**TIM Interrupt Definition**

TIM\_IT\_UPDATE  
 TIM\_IT\_CC1  
 TIM\_IT\_CC2  
 TIM\_IT\_CC3  
 TIM\_IT\_CC4  
 TIM\_IT\_COM  
 TIM\_IT\_TRIGGER  
 TIM\_IT\_BREAK

**TIM Lock level**

TIM\_LOCKLEVEL\_OFF  
 TIM\_LOCKLEVEL\_1  
 TIM\_LOCKLEVEL\_2  
 TIM\_LOCKLEVEL\_3

**TIM Master Mode Selection**

TIM\_TRGO\_RESET  
 TIM\_TRGO\_ENABLE  
 TIM\_TRGO\_UPDATE  
 TIM\_TRGO\_OC1  
 TIM\_TRGO\_OC1REF  
 TIM\_TRGO\_OC2REF  
 TIM\_TRGO\_OC3REF



TIM\_TRGO\_OC4REF

***TIM Master Slave Mode***

TIM\_MASTERSLAVEMODE\_ENABLE

TIM\_MASTERSLAVEMODE\_DISABLE

***TIM One Pulse Mode***

TIM\_OPMODE\_SINGLE

TIM\_OPMODE\_REPETITIVE

***TIM OSSI Off State Selection for Idle mode state***

TIM\_OSSI\_ENABLE

TIM\_OSSI\_DISABLE

***TIM OSSR Off State Selection for Run mode state***

TIM\_OSSR\_ENABLE

TIM\_OSSR\_DISABLE

***TIM Output Compare and PWM modes***

TIM\_OCMODE\_TIMING

TIM\_OCMODE\_ACTIVE

TIM\_OCMODE\_INACTIVE

TIM\_OCMODE\_TOGGLE

TIM\_OCMODE\_PWM1

TIM\_OCMODE\_PWM2

TIM\_OCMODE\_FORCED\_ACTIVE

TIM\_OCMODE\_FORCED\_INACTIVE

***TIM Output Compare Idle State***

TIM\_OCIDLESTATE\_SET

TIM\_OCIDLESTATE\_RESET

***TIM Complementary Output Compare Idle State***

TIM\_OCNIDLESTATE\_SET

TIM\_OCNIDLESTATE\_RESET

***TIM Complementary Output Compare Polarity***

TIM\_OCNPOLARITY\_HIGH

TIM\_OCNPOLARITY\_LOW

***TIM Complementary Output Compare State***

TIM\_OUTPUTNSTATE\_DISABLE

TIM\_OUTPUTNSTATE\_ENABLE

***TIM Output Compare Polarity***

TIM\_OCPOLARITY\_HIGH

TIM\_OCPOLARITY\_LOW

***TIM Output Compare State***

TIM\_OUTPUTSTATE\_DISABLE

TIM\_OUTPUTSTATE\_ENABLE

***TIM Output Fast State***

TIM\_OCFAST\_DISABLE

TIM\_OCFAST\_ENABLE

***TIM Private Constants***

TIM\_CCER\_CCxE\_MASK

TIM\_CCER\_CCxE\_MASK

TIM\_CCER\_CCxNE\_MASK

***TIM Private Macros***

IS\_TIM\_COUNTER\_MODE

IS\_TIM\_CLOCKDIVISION\_DIV

IS\_TIM\_PWM\_MODE

IS\_TIM\_OC\_MODE

IS\_TIM\_FAST\_STATE

IS\_TIM\_OC\_POLARITY

IS\_TIM\_OCN\_POLARITY

IS\_TIM\_OCIDLE\_STATE

IS\_TIM\_OCNIDLE\_STATE

IS\_TIM\_CHANNELS

IS\_TIM\_OPM\_CHANNELS

IS\_TIM\_COMPLEMENTARY\_CHANNELS

IS\_TIM\_IC\_POLARITY

IS\_TIM\_IC\_SELECTION

IS\_TIM\_IC\_PRESCALER

IS\_TIM\_OPM\_MODE

IS\_TIM\_ENCODER\_MODE

IS\_TIM\_DMA\_SOURCE

IS\_TIM\_EVENT\_SOURCE

IS\_TIM\_CLOCKSOURCE

IS\_TIM\_CLOCKPOLARITY

IS\_TIM\_CLOCKPRESCALER

IS\_TIM\_CLOCKFILTER

IS\_TIM\_CLEARINPUT\_SOURCE

IS\_TIM\_CLEARINPUT\_POLARITY  
IS\_TIM\_CLEARINPUT\_PRESCALER  
IS\_TIM\_CLEARINPUT\_FILTER  
IS\_TIM\_OSSR\_STATE  
IS\_TIM\_OSSI\_STATE  
IS\_TIM\_LOCK\_LEVEL  
IS\_TIM\_BREAK\_STATE  
IS\_TIM\_BREAK\_POLARITY  
IS\_TIM\_AUTOMATIC\_OUTPUT\_STATE  
IS\_TIM\_TRGO\_SOURCE  
IS\_TIM\_SLAVE\_MODE  
IS\_TIM\_MSM\_STATE  
IS\_TIM\_TRIGGER\_SELECTION  
IS\_TIM\_INTERNAL\_TRIGGEREVENT\_SELECTION  
IS\_TIM\_TRIGGERPOLARITY  
IS\_TIM\_TRIGGERPRESCALER  
IS\_TIM\_TRIGGERFILTER  
IS\_TIM\_TI1SELECTION  
IS\_TIM\_DMA\_BASE  
IS\_TIM\_DMA\_LENGTH  
IS\_TIM\_IC\_FILTER  
TIM\_SET\_ICPRESCALERVALUE

**Description:**

- Set TIM IC prescaler.

**Parameters:**

- `__HANDLE__`: TIM handle
- `__CHANNEL__`: specifies TIM Channel
- `__ICPSC__`: specifies the prescaler value.

**Return value:**

- None:

TIM\_RESET\_ICPRESCALERVALUE

**Description:**

- Reset TIM IC prescaler.

**Parameters:**

- `__HANDLE__`: TIM handle
- `__CHANNEL__`: specifies TIM Channel

**Return value:**

TIM\_SET\_CAPTUREPOLARITY

- None:

**Description:**

- Set TIM IC polarity.

**Parameters:**

- `__HANDLE__`: TIM handle
- `__CHANNEL__`: specifies TIM Channel
- `__POLARITY__`: specifies TIM Channel Polarity

**Return value:**

- None:

TIM\_RESET\_CAPTUREPOLARITY

**Description:**

- Reset TIM IC polarity.

**Parameters:**

- `__HANDLE__`: TIM handle
- `__CHANNEL__`: specifies TIM Channel

**Return value:**

- None:

***TIM Slave Mode***

TIM\_SLAVEMODE\_DISABLE

TIM\_SLAVEMODE\_RESET

TIM\_SLAVEMODE\_GATED

TIM\_SLAVEMODE\_TRIGGER

TIM\_SLAVEMODE\_EXTERNAL1

***TIM TI1 Input Selection***

TIM\_TI1SELECTION\_CH1

TIM\_TI1SELECTION\_XORCOMBINATION

***TIM Trigger Polarity***

TIM\_TRIGGERPOLARITY\_INVERTED      Polarity for ETRx trigger sources

TIM\_TRIGGERPOLARITY\_NONINVERTED      Polarity for ETRx trigger sources

TIM\_TRIGGERPOLARITY\_RISING      Polarity for TlxFPx or TI1\_ED trigger sources

TIM\_TRIGGERPOLARITY\_FALLING      Polarity for TlxFPx or TI1\_ED trigger sources

TIM\_TRIGGERPOLARITY\_BOTHEDGE      Polarity for TlxFPx or TI1\_ED trigger sources

***TIM Trigger Prescaler***

TIM\_TRIGGERPRESCALER\_DIV1      No prescaler is used

---

TIM_TRIGGERPRESCALER_DIV2	Prescaler for External ETR Trigger: Capture performed once every 2 events.
TIM_TRIGGERPRESCALER_DIV4	Prescaler for External ETR Trigger: Capture performed once every 4 events.
TIM_TRIGGERPRESCALER_DIV8	Prescaler for External ETR Trigger: Capture performed once every 8 events.

***TIM Trigger Selection***

TIM\_TS\_ITR0

TIM\_TS\_ITR1

TIM\_TS\_ITR2

TIM\_TS\_ITR3

TIM\_TS\_TI1F\_ED

TIM\_TS\_TI1FP1

TIM\_TS\_TI2FP2

TIM\_TS\_ETRF

TIM\_TS\_NONE

## 40 HAL TIM Extension Driver

### 40.1 TIMEx Firmware driver registers structures

#### 40.1.1 TIM\_HallSensor\_InitTypeDef

*TIM\_HallSensor\_InitTypeDef* is defined in the stm32f1xx\_hal\_tim\_ex.h

##### Data Fields

- *uint32\_t IC1Polarity*
- *uint32\_t IC1Prescaler*
- *uint32\_t IC1Filter*
- *uint32\_t Commutation\_Delay*

##### Field Documentation

- *uint32\_t TIM\_HallSensor\_InitTypeDef::IC1Polarity* Specifies the active edge of the input signal. This parameter can be a value of [TIM\\_Input\\_Capture\\_Polarity](#)
- *uint32\_t TIM\_HallSensor\_InitTypeDef::IC1Prescaler* Specifies the Input Capture Prescaler. This parameter can be a value of [TIM\\_Input\\_Capture\\_Prescaler](#)
- *uint32\_t TIM\_HallSensor\_InitTypeDef::IC1Filter* Specifies the input capture filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF
- *uint32\_t TIM\_HallSensor\_InitTypeDef::Commutation\_Delay* Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF

#### 40.1.2 TIM\_BreakDeadTimeConfigTypeDef

*TIM\_BreakDeadTimeConfigTypeDef* is defined in the stm32f1xx\_hal\_tim\_ex.h

##### Data Fields

- *uint32\_t OffStateRunMode*
- *uint32\_t OffStateIDLEMode*
- *uint32\_t LockLevel*
- *uint32\_t DeadTime*
- *uint32\_t BreakState*
- *uint32\_t BreakPolarity*
- *uint32\_t AutomaticOutput*

##### Field Documentation

- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::OffStateRunMode* TIM off state in run mode This parameter can be a value of [TIM\\_OSSR\\_Off\\_State\\_Selection\\_for\\_Run\\_mode\\_state](#)
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::OffStateIDLEMode* TIM off state in IDLE mode This parameter can be a value of [TIM\\_OSSI\\_Off\\_State\\_Selection\\_for\\_Idle\\_mode\\_state](#)
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::LockLevel* TIM Lock level This parameter can be a value of [TIM\\_Lock\\_level](#)

- **`uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime`** TIM dead Time This parameter can be a number between Min\_Data = 0x00 and Max\_Data = 0xFF
- **`uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState`** TIM Break State This parameter can be a value of [TIM\\_Break\\_Input\\_enable\\_disable](#)
- **`uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity`** TIM Break input polarity This parameter can be a value of [TIM\\_Break\\_Polarity](#)
- **`uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput`** TIM Automatic Output Enable state This parameter can be a value of [TIM\\_AOE\\_Bit\\_Set\\_Reset](#)

### 40.1.3 TIM\_MasterConfigTypeDef

**`TIM_MasterConfigTypeDef`** is defined in the `stm32f1xx_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 [TIM\\_Master\\_Mode\\_Selection](#)
- **`uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode`** Master/slave mode selection This parameter can be a value of [TIM\\_Master\\_Slave\\_Mode](#)

## 40.2 TIMEx Firmware driver API description

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

### 40.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
  - Output Compare
  - PWM generation (Edge and Center-aligned Mode)
  - One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

### 40.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
  - Complementary Output Compare : `HAL_TIM_OC_MspInit()`
  - Complementary PWM generation : `HAL_TIM_PWM_MspInit()`

- Complementary One-pulse mode output : HAL\_TIM\_OnePulse\_MspInit()
- Hall Sensor output : HAL\_TIMEx\_HallSensor\_MspInit()
- 2. Initialize the TIM low level resources :
  - a. Enable the TIM interface clock using \_\_HAL\_RCC\_TIMx\_CLK\_ENABLE();
  - b. TIM pins configuration
    - Enable the clock for the TIM GPIOs using the following function:  
\_\_HAL\_GPIOx\_CLK\_ENABLE();
    - Configure these TIM pins in Alternate function mode using  
HAL\_GPIO\_Init();
- 3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL\_TIM\_ConfigClockSource, the clock configuration should be done before any start function.
- 4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
  - HAL\_TIMEx\_HallSensor\_Init and HAL\_TIMEx\_ConfigCommutationEvent: to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).
- 5. Activate the TIM peripheral using one of the start functions:
  - Complementary Output Compare : HAL\_TIMEx\_OC\_N\_Start(),  
HAL\_TIMEx\_OC\_N\_Start\_DMA(), HAL\_TIMEx\_OC\_N\_Start\_IT()
  - Complementary PWM generation : HAL\_TIMEx\_PWMN\_Start(),  
HAL\_TIMEx\_PWMN\_Start\_DMA(), HAL\_TIMEx\_PWMN\_Start\_IT()
  - Complementary One-pulse mode output : HAL\_TIMEx\_OnePulseN\_Start(),  
HAL\_TIMEx\_OnePulseN\_Start\_IT()
  - Hall Sensor output : HAL\_TIMEx\_HallSensor\_Start(),  
HAL\_TIMEx\_HallSensor\_Start\_DMA(), HAL\_TIMEx\_HallSensor\_Start\_IT().

### 40.2.3 Timer Hall Sensor functions

This section provides functions allowing to:

- Initialize and configure TIM HAL Sensor.
- De-initialize TIM HAL Sensor.
- Start the Hall Sensor Interface.
- Stop the Hall Sensor Interface.
- Start the Hall Sensor Interface and enable interrupts.
- Stop the Hall Sensor Interface and disable interrupts.
- Start the Hall Sensor Interface and enable DMA transfers.
- Stop the Hall Sensor Interface and disable DMA transfers.
- [HAL\\_TIMEx\\_HallSensor\\_Init\(\)](#)
- [HAL\\_TIMEx\\_HallSensor\\_DeInit\(\)](#)
- [HAL\\_TIMEx\\_HallSensor\\_MspInit\(\)](#)
- [HAL\\_TIMEx\\_HallSensor\\_MspDeInit\(\)](#)
- [HAL\\_TIMEx\\_HallSensor\\_Start\(\)](#)
- [HAL\\_TIMEx\\_HallSensor\\_Stop\(\)](#)
- [HAL\\_TIMEx\\_HallSensor\\_Start\\_IT\(\)](#)
- [HAL\\_TIMEx\\_HallSensor\\_Stop\\_IT\(\)](#)
- [HAL\\_TIMEx\\_HallSensor\\_Start\\_DMA\(\)](#)
- [HAL\\_TIMEx\\_HallSensor\\_Stop\\_DMA\(\)](#)



## 40.2.4 Timer Complementary Output Compare functions

This section provides functions allowing to:

- Start the Complementary Output Compare/PWM.
- Stop the Complementary Output Compare/PWM.
- Start the Complementary Output Compare/PWM and enable interrupts.
- Stop the Complementary Output Compare/PWM and disable interrupts.
- Start the Complementary Output Compare/PWM and enable DMA transfers.
- Stop the Complementary Output Compare/PWM and disable DMA transfers.
- [HAL\\_TIMEx\\_OCN\\_Start\(\)](#)
- [HAL\\_TIMEx\\_OCN\\_Stop\(\)](#)
- [HAL\\_TIMEx\\_OCN\\_Start\\_IT\(\)](#)
- [HAL\\_TIMEx\\_OCN\\_Stop\\_IT\(\)](#)
- [HAL\\_TIMEx\\_OCN\\_Start\\_DMA\(\)](#)
- [HAL\\_TIMEx\\_OCN\\_Stop\\_DMA\(\)](#)

## 40.2.5 Timer Complementary PWM functions

This section provides functions allowing to:

- Start the Complementary PWM.
- Stop the Complementary PWM.
- Start the Complementary PWM and enable interrupts.
- Stop the Complementary PWM and disable interrupts.
- Start the Complementary PWM and enable DMA transfers.
- Stop the Complementary PWM and disable DMA transfers.
- Start the Complementary Input Capture measurement.
- Stop the Complementary Input Capture.
- Start the Complementary Input Capture and enable interrupts.
- Stop the Complementary Input Capture and disable interrupts.
- Start the Complementary Input Capture and enable DMA transfers.
- Stop the Complementary Input Capture and disable DMA transfers.
- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.
- [HAL\\_TIMEx\\_PWMN\\_Start\(\)](#)
- [HAL\\_TIMEx\\_PWMN\\_Stop\(\)](#)
- [HAL\\_TIMEx\\_PWMN\\_Start\\_IT\(\)](#)
- [HAL\\_TIMEx\\_PWMN\\_Stop\\_IT\(\)](#)
- [HAL\\_TIMEx\\_PWMN\\_Start\\_DMA\(\)](#)
- [HAL\\_TIMEx\\_PWMN\\_Stop\\_DMA\(\)](#)

## 40.2.6 Timer Complementary One Pulse functions

This section provides functions allowing to:

- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.

- Stop the Complementary One Pulse and disable interrupts.
- [HAL\\_TIMEx\\_OnePulseN\\_Start\(\)](#)
- [HAL\\_TIMEx\\_OnePulseN\\_Stop\(\)](#)
- [HAL\\_TIMEx\\_OnePulseN\\_Start\\_IT\(\)](#)
- [HAL\\_TIMEx\\_OnePulseN\\_Stop\\_IT\(\)](#)

## 40.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- [HAL\\_TIMEx\\_ConfigCommutationEvent\(\)](#)
- [HAL\\_TIMEx\\_ConfigCommutationEvent\\_IT\(\)](#)
- [HAL\\_TIMEx\\_ConfigCommutationEvent\\_DMA\(\)](#)
- [HAL\\_TIMEx\\_ConfigBreakDeadTime\(\)](#)
- [HAL\\_TIMEx\\_MasterConfigSynchronization\(\)](#)

## 40.2.8 Extension Callbacks functions

This section provides Extension TIM callback functions:

- Timer Commutation callback
- Timer Break callback
- [HAL\\_TIMEx\\_CommutationCallback\(\)](#)
- [HAL\\_TIMEx\\_BreakCallback\(\)](#)
- [TIMEx\\_DMACommutationCplt\(\)](#)

## 40.2.9 Extension Peripheral State functions

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

- [HAL\\_TIMEx\\_HallSensor\\_GetState\(\)](#)

## 40.2.10 HAL\_TIMEx\_HallSensor\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init</b> (TIM_HandleTypeDef * htim, TIM_HallSensor_InitTypeDef * sConfig)
Function Description	Initializes the TIM Hall Sensor Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : TIM Encoder Interface handle</li> <li>• <b>sConfig</b> : TIM Hall Sensor configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 40.2.11 HAL\_TIMEx\_HallSensor\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes the TIM Hall Sensor interface.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Hall Sensor handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 40.2.12 HAL\_TIMEx\_HallSensor\_MspInit

Function Name	<b>void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Hall Sensor MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 40.2.13 HAL\_TIMEx\_HallSensor\_MspDeInit

Function Name	<b>void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM Hall Sensor MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 40.2.14 HAL\_TIMEx\_HallSensor\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)</b>
Function Description	Starts the TIM Hall Sensor Interface.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Hall Sensor handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 40.2.15 HAL\_TIMEx\_HallSensor\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)</b>
Function Description	Stops the TIM Hall sensor Interface.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM Hall Sensor handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 40.2.16 HAL\_TIMEx\_HallSensor\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT</b> (TIM_HandleTypeDef * htim)
Function Description	Starts the TIM Hall Sensor Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b>: : TIM Hall Sensor handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 40.2.17 HAL\_TIMEx\_HallSensor\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT</b> (TIM_HandleTypeDef * htim)
Function Description	Stops the TIM Hall Sensor Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b>: : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 40.2.18 HAL\_TIMEx\_HallSensor\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA</b> (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM Hall Sensor Interface in DMA mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b>: : TIM Hall Sensor handle</li><li>• <b>pData</b>: : The destination Buffer address.</li><li>• <b>Length</b>: : The length of data to be transferred from TIM peripheral to memory.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 40.2.19 HAL\_TIMEx\_HallSensor\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA</b> (TIM_HandleTypeDef * htim)
Function Description	Stops the TIM Hall Sensor Interface in DMA mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b>: : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 40.2.20 HAL\_TIMEx\_OC\_N\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OC_N_Start</b> (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Output Compare signal generation on the

complementary output.

- |               |  |
|---------------|--|
| Parameters    | <ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM Output Compare handle</li> <li>• <b>Channel:</b> : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> |
| Return values | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>   |

#### 40.2.21 HAL\_TIMEx\_OCN\_Stop

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</b>   |
| Function Description | Stops the TIM Output Compare signal generation on the complementary output.  |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM handle</li> <li>• <b>Channel:</b> : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>   |

#### 40.2.22 HAL\_TIMEx\_OCN\_Start\_IT

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>   |
| Function Description | Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM OC handle</li> <li>• <b>Channel:</b> : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected</li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>   |

#### 40.2.23 HAL\_TIMEx\_OCN\_Stop\_IT

- |                      |   |
|----------------------|---|
| Function Name        | <b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>   |
| Function Description | Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>htim:</b> : TIM Output Compare handle</li> <li>• <b>Channel:</b> : TIM Channel to be disabled This parameter can</li> </ul> |

be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

## Return values

- HAL status

#### 40.2.24 HAL\_TIMEx\_OCN\_Start\_DMA

## Function Name

**HAL\_StatusTypeDef HAL\_TIMEx\_OCN\_Start\_DMA**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel, uint32\_t \* pData, uint16\_t Length)

## Function Description

Starts the TIM Output Compare signal generation in DMA mode on the complementary output.

## Parameters

- **htim**: : TIM Output Compare handle
- **Channel**: : TIM Channel to be enabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected
- **pData**: : The source Buffer address.
- **Length**: : The length of data to be transferred from memory to TIM peripheral

## Return values

- HAL status

#### 40.2.25 HAL\_TIMEx\_OCN\_Stop\_DMA

## Function Name

**HAL\_StatusTypeDef HAL\_TIMEx\_OCN\_Stop\_DMA**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)

## Function Description

Stops the TIM Output Compare signal generation in DMA mode on the complementary output.

## Parameters

- **htim**: : TIM Output Compare handle
- **Channel**: : TIM Channel to be disabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

## Return values

- HAL status

#### 40.2.26 HAL\_TIMEx\_PWMN\_Start

## Function Name

**HAL\_StatusTypeDef HAL\_TIMEx\_PWMN\_Start**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)

## Function Description

Starts the PWM signal generation on the complementary output.

## Parameters

- **htim**: : TIM handle
- **Channel**: : TIM Channel to be enabled This parameter can be

one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

#### 40.2.27 HAL\_TIMEx\_PWMN\_Stop

**Function Name** `HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)`

**Function Description** Stops the PWM signal generation on the complementary output.

**Parameters**

- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

#### 40.2.28 HAL\_TIMEx\_PWMN\_Start\_IT

**Function Name** `HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)`

**Function Description** Starts the PWM signal generation in interrupt mode on the complementary output.

**Parameters**

- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

#### 40.2.29 HAL\_TIMEx\_PWMN\_Stop\_IT

**Function Name** `HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)`

**Function Description** Stops the PWM signal generation in interrupt mode on the complementary output.

**Parameters**

- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

#### 40.2.30 HAL\_TIMEx\_PWMN\_Start\_DMA

**Function Name** **HAL\_StatusTypeDef HAL\_TIMEx\_PWMN\_Start\_DMA (TIM\_HandleTypeDef \* htim, uint32\_t Channel, uint32\_t \* pData, uint16\_t Length)**

**Function Description** Starts the TIM PWM signal generation in DMA mode on the complementary output.

**Parameters**

- **htim**: : TIM handle
- **Channel**: : TIM Channel to be enabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected TIM\_CHANNEL\_3: TIM Channel 3 selected TIM\_CHANNEL\_4: TIM Channel 4 selected
- **pData**: : The source Buffer address.
- **Length**: : The length of data to be transferred from memory to TIM peripheral

Return values

- HAL status

#### 40.2.31 HAL\_TIMEx\_PWMN\_Stop\_DMA

**Function Name** **HAL\_StatusTypeDef HAL\_TIMEx\_PWMN\_Stop\_DMA (TIM\_HandleTypeDef \* htim, uint32\_t Channel)**

**Function Description** Stops the TIM PWM signal generation in DMA mode on the complementary output.

**Parameters**

- **htim**: : TIM handle
- **Channel**: : TIM Channel to be disabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected TIM\_CHANNEL\_3: TIM Channel 3 selected TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

#### 40.2.32 HAL\_TIMEx\_OnePulseN\_Start

**Function Name** **HAL\_StatusTypeDef HAL\_TIMEx\_OnePulseN\_Start (TIM\_HandleTypeDef \* htim, uint32\_t OutputChannel)**

**Function Description** Starts the TIM One Pulse signal generation on the complementary output.

**Parameters**

- **htim**: : TIM One Pulse handle
- **OutputChannel**: : TIM Channel to be enabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected



Return values

- HAL status

### 40.2.33 HAL\_TIMEx\_OnePulseN\_Stop

**Function Name** `HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)`

**Function Description** Stops the TIM One Pulse signal generation on the complementary output.

**Parameters**

- **htim**: : TIM One Pulse handle
- **OutputChannel**: : TIM Channel to be disabled This parameter can be one of the following values:  
TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected

**Return values**

- HAL status

### 40.2.34 HAL\_TIMEx\_OnePulseN\_Start\_IT

**Function Name** `HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)`

**Function Description** Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.

**Parameters**

- **htim**: : TIM One Pulse handle
- **OutputChannel**: : TIM Channel to be enabled This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected TIM\_CHANNEL\_2: TIM Channel 2 selected

**Return values**

- HAL status

### 40.2.35 HAL\_TIMEx\_OnePulseN\_Stop\_IT

**Function Name** `HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)`

**Function Description** Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.

**Parameters**

- **htim**: : TIM One Pulse handle
- **OutputChannel**: : TIM Channel to be disabled This parameter can be one of the following values:  
TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected

**Return values**

- HAL status

### 40.2.36 HAL\_TIMEx\_ConfigCommutationEvent

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent</b> (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
Function Description	Configure the TIM commutation event sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : TIM handle</li> <li>• <b>InputTrigger</b> : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values: TIM_TS_ITR0: Internal trigger 0 selected TIM_TS_ITR1: Internal trigger 1 selected TIM_TS_ITR2: Internal trigger 2 selected TIM_TS_ITR3: Internal trigger 3 selected TIM_TS_NONE: No trigger is needed</li> <li>• <b>CommutationSource</b> : the Commutation Event source This parameter can be one of the following values: TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.</li> </ul>

#### 40.2.37 HAL\_TIMEx\_ConfigCommutationEvent\_IT

Function Name	<b>HAL_StatusTypeDef</b> <b>HAL_TIMEx_ConfigCommutationEvent_IT</b> (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
Function Description	Configure the TIM commutation event sequence with interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : TIM handle</li> <li>• <b>InputTrigger</b> : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values: TIM_TS_ITR0: Internal trigger 0 selected TIM_TS_ITR1: Internal trigger 1 selected TIM_TS_ITR2: Internal trigger 2 selected TIM_TS_ITR3: Internal trigger 3 selected TIM_TS_NONE: No trigger is needed</li> <li>• <b>CommutationSource</b> : the Commutation Event source This parameter can be one of the following values: TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• : this function is mandatory to use the commutation event in</li> </ul>

order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

#### 40.2.38 HAL\_TIMEx\_ConfigCommutationEvent\_DMA

Function Name	<b>HAL_StatusTypeDef</b> <b>HAL_TIMEx_ConfigCommutationEvent_DMA</b> <b>(TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)</b>
Function Description	Configure the TIM commutation event sequence with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> <li>• <b>InputTrigger</b>: : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values: TIM_TS_ITR0: Internal trigger 0 selected TIM_TS_ITR1: Internal trigger 1 selected TIM_TS_ITR2: Internal trigger 2 selected TIM_TS_ITR3: Internal trigger 3 selected TIM_TS_NONE: No trigger is needed</li> <li>• <b>CommutationSource</b>: : the Commutation Event source This parameter can be one of the following values: TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.</li> <li>• : The user should configure the DMA in his own software, in This function only the COMDE bit is set</li> </ul>

#### 40.2.39 HAL\_TIMEx\_ConfigBreakDeadTime

Function Name	<b>HAL_StatusTypeDef</b> <b>HAL_TIMEx_ConfigBreakDeadTime</b> <b>(TIM_HandleTypeDef * htim,</b> <b>TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)</b>
Function Description	Configures the Break feature, dead time, Lock level, OSSR/OSSR State and the AOE(automatic output enable).
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: : TIM handle</li> <li>• <b>sBreakDeadTimeConfig</b>: : pointer to a</li> </ul>

TIM\_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.

Return values

- HAL status

#### 40.2.40 HAL\_TIMEx\_MasterConfigSynchronization

**Function Name** **HAL\_StatusTypeDef HAL\_TIMEx\_MasterConfigSynchronization (TIM\_HandleTypeDef \* htim, TIM\_MasterConfigTypeDef \* sMasterConfig)**

**Function Description** Configures the TIM in master mode.

**Parameters**

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

**Return values**

- HAL status

#### 40.2.41 HAL\_TIMEx\_CommutationCallback

**Function Name** **void HAL\_TIMEx\_CommutationCallback (TIM\_HandleTypeDef \* htim)**

**Function Description** Hall commutation changed callback in non blocking mode.

**Parameters**

- **htim**: : TIM handle

**Return values**

- None

#### 40.2.42 HAL\_TIMEx\_BreakCallback

**Function Name** **void HAL\_TIMEx\_BreakCallback (TIM\_HandleTypeDef \* htim)**

**Function Description** Hall Break detection callback in non blocking mode.

**Parameters**

- **htim**: : TIM handle

**Return values**

- None

#### 40.2.43 TIMEx\_DMACommutationCplt

**Function Name** **void TIMEx\_DMACommutationCplt (DMA\_HandleTypeDef \* hdma)**

**Function Description** TIM DMA Commutation callback.

**Parameters**

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

**Return values**

- None

#### 40.2.44 HAL\_TIMEx\_HallSensor\_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Hall Sensor interface state.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b>: : TIM Hall Sensor handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL state</li></ul>

### 40.3 TIMEx Firmware driver defines

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

#### 40.3.1 TIMEx

TIMEx

*TIMEx Clock Filter*

IS\_TIM\_DEADTIME BreakDead Time

## 41 HAL UART Generic Driver

### 41.1 UART Firmware driver registers structures

#### 41.1.1 UART\_InitTypeDef

*UART\_InitTypeDef* is defined in the stm32f1xx\_hal\_uart.h

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t HwFlowCtl*
- *uint32\_t OverSampling*

##### Field Documentation

- *uint32\_t UART\_InitTypeDef::BaudRate* This member configures the UART communication baud rate. The baud rate is computed using the following formula:
  - IntegerDivider = ((PCLKx) / (16 \* (huart->Init.BaudRate)))
  - FractionalDivider = ((IntegerDivider - ((uint32\_t) IntegerDivider)) \* 16) + 0.5
- *uint32\_t UART\_InitTypeDef::WordLength* Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [UART\\_Word\\_Length](#)
- *uint32\_t UART\_InitTypeDef::StopBits* Specifies the number of stop bits transmitted. This parameter can be a value of [UART\\_Stop\\_Bits](#)
- *uint32\_t UART\_InitTypeDef::Parity* Specifies the parity mode. This parameter can be a value of [UART\\_Parity](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- *uint32\_t UART\_InitTypeDef::Mode* Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [UART\\_Mode](#)
- *uint32\_t UART\_InitTypeDef::HwFlowCtl* Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of [UART\\_Hardware\\_Flow\\_Control](#)
- *uint32\_t UART\_InitTypeDef::OverSampling* Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to fPCLK/8). This parameter can be a value of [UART\\_Over\\_Sampling](#). This feature is not available on STM32F1xx family, so OverSampling parameter should always be set to 16.

#### 41.1.2 UART\_HandleTypeDef

*UART\_HandleTypeDef* is defined in the stm32f1xx\_hal\_uart.h

##### Data Fields

- *USART\_TypeDef \* Instance*
- *UART\_InitTypeDef Init*

- `uint8_t *pTxBuffPtr`
- `uint16_t TxXferSize`
- `uint16_t TxXferCount`
- `uint8_t *pRxBuffPtr`
- `uint16_t RxXferSize`
- `uint16_t RxXferCount`
- `DMA_HandleTypeDef *hdmatx`
- `DMA_HandleTypeDef *hdmarx`
- `HAL_LockTypeDef Lock`
- `__IO HAL_UART_StateTypeDef State`
- `__IO uint32_t ErrorCode`

#### Field Documentation

- **`USART_TypeDef* UART_HandleTypeDef::Instance`** UART registers base address
- **`UART_InitTypeDef UART_HandleTypeDef::Init`** UART communication parameters
- **`uint8_t* UART_HandleTypeDef::pTxBuffPtr`** Pointer to UART Tx transfer Buffer
- **`uint16_t UART_HandleTypeDef::TxXferSize`** UART Tx Transfer size
- **`uint16_t UART_HandleTypeDef::TxXferCount`** UART Tx Transfer Counter
- **`uint8_t* UART_HandleTypeDef::pRxBuffPtr`** Pointer to UART Rx transfer Buffer
- **`uint16_t UART_HandleTypeDef::RxXferSize`** UART Rx Transfer size
- **`uint16_t UART_HandleTypeDef::RxXferCount`** UART Rx Transfer Counter
- **`DMA_HandleTypeDef* UART_HandleTypeDef::hdmatx`** UART Tx DMA Handle parameters
- **`DMA_HandleTypeDef* UART_HandleTypeDef::hdmarx`** UART Rx DMA Handle parameters
- **`HAL_LockTypeDef UART_HandleTypeDef::Lock`** Locking object
- **`__IO HAL_UART_StateTypeDef UART_HandleTypeDef::State`** UART communication state
- **`__IO uint32_t UART_HandleTypeDef::ErrorCode`** UART Error code

## 41.2 UART Firmware driver API description

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

### 41.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a `UART_HandleTypeDef` handle structure.
2. Initialize the UART low level resources by implementing the `HAL_UART_MspInit()` API:
  - a. Enable the USARTx interface clock.
  - b. UART pins configuration:
    - Enable the clock for the UART GPIOs.
    - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
  - c. NVIC configuration if you need to use interrupt process (`HAL_UART_Transmit_IT()` and `HAL_UART_Receive_IT()` APIs):
    - Configure the USARTx interrupt priority.
    - Enable the NVIC USART IRQ handle.

- d. DMA Configuration if you need to use DMA process (HAL\_UART\_Transmit\_DMA() and HAL\_UART\_Receive\_DMA() APIs):
  - Declare a DMA handle structure for the Tx/Rx channel.
  - Enable the DMAx interface clock.
  - Configure the declared DMA handle structure with the required Tx/Rx parameters.
  - Configure the DMA Tx/Rx channel.
  - Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
  - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the huart Init structure.
4. For the UART asynchronous mode, initialize the UART registers by calling the HAL\_UART\_Init() API.
5. For the UART Half duplex mode, initialize the UART registers by calling the HAL\_HalfDuplex\_Init() API.
6. For the LIN mode, initialize the UART registers by calling the HAL\_LIN\_Init() API.
7. For the Multi-Processor mode, initialize the UART registers by calling the HAL\_MultiProcessor\_Init() API.



The specific UART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros `__HAL_UART_ENABLE_IT()` and `__HAL_UART_DISABLE_IT()` inside the transmit and receive process.



These APIs (HAL\_UART\_Init() and HAL\_HalfDuplex\_Init()) configure also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL\_UART\_MspInit() API.

Three operation modes are available within this driver :

### Polling mode IO operation

- Send an amount of data in blocking mode using HAL\_UART\_Transmit()
- Receive an amount of data in blocking mode using HAL\_UART\_Receive()

### Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL\_UART\_Transmit\_IT()
- At transmission end of transfer HAL\_UART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_UART\_Receive\_IT()
- At reception end of transfer HAL\_UART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxCpltCallback
- In case of transfer Error, HAL\_UART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_UART\_ErrorCallback



## DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using `HAL_UART_Transmit_DMA()`
- At transmission end of half transfer `HAL_UART_TxHalfCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_UART_TxHalfCpltCallback`
- At transmission end of transfer `HAL_UART_TxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_UART_TxCpltCallback`
- Receive an amount of data in non blocking mode (DMA) using `HAL_UART_Receive_DMA()`
- At reception end of half transfer `HAL_UART_RxHalfCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_UART_RxHalfCpltCallback`
- At reception end of transfer `HAL_UART_RxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_UART_RxCpltCallback`
- In case of transfer Error, `HAL_UART_ErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_UART_ErrorCallback`
- Pause the DMA Transfer using `HAL_UART_DMAPause()`
- Resume the DMA Transfer using `HAL_UART_DMAResume()`
- Stop the DMA Transfer using `HAL_UART_DMAStop()`

## UART HAL driver macros list

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

- `__HAL_UART_ENABLE`: Enable the UART peripheral
- `__HAL_UART_DISABLE`: Disable the UART peripheral
- `__HAL_UART_GET_FLAG`: Check whether the specified UART flag is set or not
- `__HAL_UART_CLEAR_FLAG`: Clear the specified UART pending flag
- `__HAL_UART_ENABLE_IT`: Enable the specified UART interrupt
- `__HAL_UART_DISABLE_IT`: Disable the specified UART interrupt
- `__HAL_UART_GET_IT_SOURCE`: Check whether the specified UART interrupt has occurred or not



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

## 41.2.2 Initialization and Configuration functions

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

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), the possible UART frame formats are as listed in [Table 22: "UART frame formats"](#).

- Hardware flow control
- Receiver/transmitter modes

Table 22: UART frame formats

M bit	PCE bit	UART frame
0	0	SB   8 bit data   STB
0	1	SB   7 bit data   PB   STB
1	0	SB   9 bit data   STB
1	1	SB   8 bit data   PB   STB

The HAL\_UART\_Init(), HAL\_HalfDuplex\_Init(), HAL\_LIN\_Init() and HAL\_MultiProcessor\_Init() APIs follow respectively the UART asynchronous, UART Half duplex, LIN and Multi-Processor configuration procedures (details for the procedures are available in reference manuals (RM0008 for STM32F10Xxx MCUs and RM0041 for STM32F100xx MCUs)).

- [HAL\\_UART\\_Init\(\)](#)
- [HAL\\_HalfDuplex\\_Init\(\)](#)
- [HAL\\_LIN\\_Init\(\)](#)
- [HAL\\_MultiProcessor\\_Init\(\)](#)
- [HAL\\_UART\\_DeInit\(\)](#)
- [HAL\\_UART\\_MspInit\(\)](#)
- [HAL\\_UART\\_MspDeInit\(\)](#)

### 41.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the UART asynchronous and Half duplex data transfers.

1. There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - Non blocking mode: The communication is performed using Interrupts or DMA, these APIs return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_UART\_TxCpltCallback(), HAL\_UART\_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or receive process. The HAL\_UART\_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode APIs are:
  - HAL\_UART\_Transmit()
  - HAL\_UART\_Receive()
3. Non Blocking mode APIs with Interrupt are:
  - HAL\_UART\_Transmit\_IT()
  - HAL\_UART\_Receive\_IT()
  - HAL\_UART\_IRQHandler()
4. Non Blocking mode functions with DMA are:
  - HAL\_UART\_Transmit\_DMA()
  - HAL\_UART\_Receive\_DMA()
  - HAL\_UART\_DMAPause()
  - HAL\_UART\_DMAResume()

- HAL\_UART\_DMAStop()
- 5. A set of Transfer Complete Callbacks are provided in non blocking mode:
  - HAL\_UART\_TxCpltCallback()
  - HAL\_UART\_TxCpltCallback()
  - HAL\_UART\_RxCpltCallback()
  - HAL\_UART\_RxCpltCallback()
  - HAL\_UART\_ErrorCallback()



In the Half duplex communication, it is forbidden to run the transmit and receive process in parallel, the UART state HAL\_UART\_STATE\_BUSY\_TX\_RX can't be useful.

- [\*HAL\\_UART\\_Transmit\(\)\*](#)
- [\*HAL\\_UART\\_Receive\(\)\*](#)
- [\*HAL\\_UART\\_Transmit\\_IT\(\)\*](#)
- [\*HAL\\_UART\\_Receive\\_IT\(\)\*](#)
- [\*HAL\\_UART\\_Transmit\\_DMA\(\)\*](#)
- [\*HAL\\_UART\\_Receive\\_DMA\(\)\*](#)
- [\*HAL\\_UART\\_DMABase\(\)\*](#)
- [\*HAL\\_UART\\_DMABaseResume\(\)\*](#)
- [\*HAL\\_UART\\_DMAStop\(\)\*](#)
- [\*HAL\\_UART\\_IRQHandler\(\)\*](#)
- [\*HAL\\_UART\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_ErrorCallback\(\)\*](#)

#### 41.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART:

- HAL\_LIN\_SendBreak() API can be helpful to transmit the break character.
- HAL\_MultiProcessor\_EnterMuteMode() API can be helpful to enter the UART in mute mode.
- HAL\_MultiProcessor\_ExitMuteMode() API can be helpful to exit the UART mute mode by software.
- HAL\_HalfDuplex\_EnableTransmitter() API to enable the UART transmitter and disables the UART receiver in Half Duplex mode
- HAL\_HalfDuplex\_EnableReceiver() API to enable the UART receiver and disables the UART transmitter in Half Duplex mode
- [\*HAL\\_LIN\\_SendBreak\(\)\*](#)
- [\*HAL\\_MultiProcessor\\_EnterMuteMode\(\)\*](#)
- [\*HAL\\_MultiProcessor\\_ExitMuteMode\(\)\*](#)
- [\*HAL\\_HalfDuplex\\_EnableTransmitter\(\)\*](#)
- [\*HAL\\_HalfDuplex\\_EnableReceiver\(\)\*](#)

#### 41.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of UART communication process, return Peripheral Errors occurred during communication process

- HAL\_UART\_GetState() API can be helpful to check in run-time the state of the UART peripheral.
- HAL\_UART\_GetError() check in run-time errors that could be occurred during communication.
- [HAL\\_UART\\_GetState\(\)](#)
- [HAL\\_UART\\_GetError\(\)](#)

#### 41.2.6 HAL\_UART\_Init

Function Name	<b>HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)</b>
Function Description	Initializes the UART mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

#### 41.2.7 HAL\_HalfDuplex\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)</b>
Function Description	Initializes the half-duplex mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

#### 41.2.8 HAL\_LIN\_Init

Function Name	<b>HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)</b>
Function Description	Initializes the LIN mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li><li>• <b>BreakDetectLength:</b> Specifies the LIN break detection length. This parameter can be one of the following values: UART_LINBREAKDETECTLENGTH_10B: 10-bit break detection UART_LINBREAKDETECTLENGTH_11B: 11-bit break detection</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

**41.2.9 HAL\_MultiProcessor\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)</b>
Function Description	Initializes the Multi-Processor mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> <li>• <b>Address:</b> UART node address</li> <li>• <b>WakeUpMethod:</b> specifies the UART wakeup method. This parameter can be one of the following values: UART_WAKEUPMETHOD_IDLELINE: Wakeup by an idle line detection UART_WAKEUPMETHOD_ADDRESSMARK: Wakeup by an address mark</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**41.2.10 HAL\_UART\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)</b>
Function Description	DeInitializes the UART peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**41.2.11 HAL\_UART\_MspInit**

Function Name	<b>void HAL_UART_MspInit (UART_HandleTypeDef * huart)</b>
Function Description	UART MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**41.2.12 HAL\_UART\_MspDeInit**

Function Name	<b>void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)</b>
Function Description	UART MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> Pointer to a UART_HandleTypeDef structure that</li> </ul>

contains the configuration information for the specified UART module.

Return values

- None

### 41.2.13 HAL\_UART\_Transmit

**Function Name** **HAL\_StatusTypeDef HAL\_UART\_Transmit**  
(UART\_HandleTypeDef \* huart, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)

**Function Description** Sends an amount of data in blocking mode.

**Parameters**

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be sent
- **Timeout**: Timeout duration

Return values

- HAL status

### 41.2.14 HAL\_UART\_Receive

**Function Name** **HAL\_StatusTypeDef HAL\_UART\_Receive**  
(UART\_HandleTypeDef \* huart, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)

**Function Description** Receives an amount of data in blocking mode.

**Parameters**

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be received
- **Timeout**: Timeout duration

Return values

- HAL status

### 41.2.15 HAL\_UART\_Transmit\_IT

**Function Name** **HAL\_StatusTypeDef HAL\_UART\_Transmit\_IT**  
(UART\_HandleTypeDef \* huart, uint8\_t \* pData, uint16\_t Size)

**Function Description** Sends an amount of data in non blocking mode.

**Parameters**

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be sent

Return values

- HAL status

**41.2.16 HAL\_UART\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive_IT</b> (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Receives an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b>: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**41.2.17 HAL\_UART\_Transmit\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit_DMA</b> (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Sends an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b>: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**41.2.18 HAL\_UART\_Receive\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive_DMA</b> (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Receives an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b>: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)</li> </ul>

**41.2.19 HAL\_UART\_DMAPause**

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAPause</b> (UART_HandleTypeDef * huart)
---------------	--

Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b>: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

#### 41.2.20 HAL\_UART\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)</b>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b>: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

#### 41.2.21 HAL\_UART\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b>: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

#### 41.2.22 HAL\_UART\_IRQHandler

Function Name	<b>void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)</b>
Function Description	This function handles UART interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b>: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

#### 41.2.23 HAL\_UART\_TxCpltCallback

Function Name	<b>void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b>: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART</li></ul>



module.

Return values

- None

#### 41.2.24 HAL\_UART\_TxHalfCpltCallback

Function Name **void HAL\_UART\_TxHalfCpltCallback (UART\_HandleTypeDef \*huart)**

Function Description Tx Half Transfer completed callbacks.

Parameters

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- None

#### 41.2.25 HAL\_UART\_RxCpltCallback

Function Name **void HAL\_UART\_RxCpltCallback (UART\_HandleTypeDef \*huart)**

Function Description Rx Transfer completed callbacks.

Parameters

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- None

#### 41.2.26 HAL\_UART\_RxHalfCpltCallback

Function Name **void HAL\_UART\_RxHalfCpltCallback (UART\_HandleTypeDef \*huart)**

Function Description Rx Half Transfer completed callbacks.

Parameters

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- None

#### 41.2.27 HAL\_UART\_ErrorCallback

Function Name **void HAL\_UART\_ErrorCallback (UART\_HandleTypeDef \*huart)**

Function Description UART error callbacks.

Parameters

- **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values • None

#### 41.2.28 HAL\_LIN\_SendBreak

Function Name **HAL\_StatusTypeDef HAL\_LIN\_SendBreak (UART\_HandleTypeDef \* huart)**

Function Description Transmits break characters.

Parameters • **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values • HAL status

#### 41.2.29 HAL\_MultiProcessor\_EnterMuteMode

Function Name **HAL\_StatusTypeDef HAL\_MultiProcessor\_EnterMuteMode (UART\_HandleTypeDef \* huart)**

Function Description Enters the UART in mute mode.

Parameters • **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values • HAL status

#### 41.2.30 HAL\_MultiProcessor\_ExitMuteMode

Function Name **HAL\_StatusTypeDef HAL\_MultiProcessor\_ExitMuteMode (UART\_HandleTypeDef \* huart)**

Function Description Exits the UART mute mode: wake up software.

Parameters • **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values • HAL status

#### 41.2.31 HAL\_HalfDuplex\_EnableTransmitter

Function Name **HAL\_StatusTypeDef HAL\_HalfDuplex\_EnableTransmitter (UART\_HandleTypeDef \* huart)**

Function Description Enables the UART transmitter and disables the UART receiver.

Parameters • **huart**: Pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values • HAL status

**41.2.32 HAL\_HalfDuplex\_EnableReceiver**

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)</b>
Function Description	Enables the UART receiver and disables the UART transmitter.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**41.2.33 HAL\_UART\_GetState**

Function Name	<b>HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)</b>
Function Description	Returns the UART state.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

**41.2.34 HAL\_UART\_GetError**

Function Name	<b>uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)</b>
Function Description	Return the UART error code.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>UART Error Code</li> </ul>

**41.3 UART Firmware driver defines**

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

**41.3.1 UART**

UART

**UART Error Codes**

HAL_UART_ERROR_NONE	No error
HAL_UART_ERROR_PE	Parity error
HAL_UART_ERROR_NE	Noise error
HAL_UART_ERROR_FE	frame error
HAL_UART_ERROR_ORE	Overrun error
HAL_UART_ERROR_DMA	DMA transfer error

**UART Exported Macros****\_\_HAL\_UART\_RESET\_HANDLE\_STATE****Description:**

- Reset UART handle state.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

**Return value:**

- None:

**\_\_HAL\_UART\_FLUSH\_DRREGISTER****Description:**

- Flush the UART DR register.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

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

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

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - **UART\_FLAG\_CTS**: CTS Change flag (not available for UART4 and UART5)
  - **UART\_FLAG\_LBD**: LIN Break detection flag
  - **UART\_FLAG\_TXE**: Transmit data register empty flag
  - **UART\_FLAG\_TC**: Transmission Complete flag
  - **UART\_FLAG\_RXNE**: Receive data register not empty flag
  - **UART\_FLAG\_IDLE**: Idle Line detection flag
  - **UART\_FLAG\_ORE**: OverRun Error flag
  - **UART\_FLAG\_NE**: Noise Error

- flag
- UART\_FLAG\_FE: Framing Error flag
- UART\_FLAG\_PE: Parity Error flag

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

**Description:**

- Clear the specified UART pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be any combination of the following values:
  - UART\_FLAG\_CTS: CTS Change flag (not available for UART4 and UART5).
  - UART\_FLAG\_LBD: LIN Break detection flag.
  - UART\_FLAG\_TC: Transmission Complete flag.
  - UART\_FLAG\_RXNE: Receive data register not empty flag.

**Return value:**

- None:

**Description:**

- Clear the UART PE pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

**Return value:**

- None:

**Description:**

- Clear the UART FE pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the UART

\_\_HAL\_UART\_CLEAR\_FLAG

\_\_HAL\_UART\_CLEAR\_PEFLAG

\_\_HAL\_UART\_CLEAR\_FEFLAG

Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

**Return value:**

- None:

**Description:**

- Clear the UART NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

**Return value:**

- None:

**Description:**

- Clear the UART ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

**Return value:**

- None:

**Description:**

- Clear the UART IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

**Return value:**

- None:

**Description:**

- Enable the specified UART interrupt.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral

`__HAL_UART_CLEAR_NEFLAG`

`__HAL_UART_CLEAR_OREFLAG`

`__HAL_UART_CLEAR_IDLEFLAG`

`__HAL_UART_ENABLE_IT`

(USART, UART availability and x,y values depending on device).

- **\_\_INTERRUPT\_\_**: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - **UART\_IT\_CTS**: CTS change interrupt
  - **UART\_IT\_LBD**: LIN Break detection interrupt
  - **UART\_IT\_TXE**: Transmit Data Register empty interrupt
  - **UART\_IT\_TC**: Transmission complete interrupt
  - **UART\_IT\_RXNE**: Receive Data register not empty interrupt
  - **UART\_IT\_IDLE**: Idle line detection interrupt
  - **UART\_IT\_PE**: Parity Error interrupt
  - **UART\_IT\_ERR**: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

**Description:**

- Disable the specified UART interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- **\_\_INTERRUPT\_\_**: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - **UART\_IT\_CTS**: CTS change interrupt
  - **UART\_IT\_LBD**: LIN Break detection interrupt
  - **UART\_IT\_TXE**: Transmit Data Register empty interrupt
  - **UART\_IT\_TC**: Transmission complete interrupt
  - **UART\_IT\_RXNE**: Receive Data register not empty interrupt
  - **UART\_IT\_IDLE**: Idle line detection interrupt
  - **UART\_IT\_PE**: Parity Error

**\_\_HAL\_UART\_DISABLE\_IT**

`__HAL_UART_GET_IT_SOURCE`

- interrupt
  - UART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- `__IT__`: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - UART\_IT\_CTS: CTS change interrupt (not available for UART4 and UART5)
  - UART\_IT\_LBD: LIN Break detection interrupt
  - UART\_IT\_TXE: Transmit Data Register empty interrupt
  - UART\_IT\_TC: Transmission complete interrupt
  - UART\_IT\_RXNE: Receive Data register not empty interrupt
  - UART\_IT\_IDLE: Idle line detection interrupt
  - UART\_IT\_ERR: Error interrupt

**Return value:**

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

`__HAL_UART_HWCONTROL_CTS_ENABLE`**Description:**

- Enable CTS flow control This macro allows to enable CTS hardware flow control for a given UART instance, without need to call

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).



**\_\_HAL\_UART\_HWCONTROL\_CTS\_DISABLE****Return value:**

- None:

**Description:**

- Disable CTS flow control This macro allows to disable CTS hardware flow control for a given UART instance, without need to call

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

**Return value:**

- None:

**\_\_HAL\_UART\_HWCONTROL\_RTS\_ENABLE****Description:**

- Enable RTS flow control This macro allows to enable RTS hardware flow control for a given UART instance, without need to call

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

**Return value:**

- None:

**\_\_HAL\_UART\_HWCONTROL\_RTS\_DISABLE****Description:**

- Disable RTS flow control This macro allows to disable RTS hardware flow control for a given UART instance, without need to call

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

\_\_HAL\_UART\_ENABLE

**Return value:**

- None:

**Description:**

- Enable UART.

**Parameters:**

- \_\_HANDLE\_\_: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None:

**Description:**

- Disable UART UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None:

\_\_HAL\_UART\_DISABLE

**UART Flags**

UART\_FLAG\_CTS

UART\_FLAG\_LBD

UART\_FLAG\_TXE

UART\_FLAG\_TC

UART\_FLAG\_RXNE

UART\_FLAG\_IDLE

UART\_FLAG\_ORE

UART\_FLAG\_NE

UART\_FLAG\_FE

UART\_FLAG\_PE

**UART Hardware Flow Control**

UART\_HWCONTROL\_NONE

UART\_HWCONTROL\_RTS

UART\_HWCONTROL\_CTS

UART\_HWCONTROL\_RTS\_CTS

**UART Interrupt Definitions**

UART\_IT\_PE

UART\_IT\_TXE

UART\_IT\_TC

UART\_IT\_RXNE

UART\_IT\_IDLE

UART\_IT\_LBD

UART\_IT\_CTS

UART\_IT\_ERR

***UART LIN Break Detection Length***

UART\_LINBREAKDETECTLENGTH\_10B

UART\_LINBREAKDETECTLENGTH\_11B

***UART Transfer Mode***

UART\_MODE\_RX

UART\_MODE\_TX

UART\_MODE\_TX\_RX

***UART Over Sampling***

UART\_OVERSAMPLING\_16

***UART Parity***

UART\_PARITY\_NONE

UART\_PARITY\_EVEN

UART\_PARITY\_ODD

***UART Private Macros***

UART\_CR1\_REG\_INDEX

UART\_CR2\_REG\_INDEX

UART\_CR3\_REG\_INDEX

UART\_DIV\_SAMPLING16

UART\_DIVMANT\_SAMPLING16

UART\_DIVFRAQ\_SAMPLING16

UART\_BRR\_SAMPLING16

IS\_UART\_WORD\_LENGTH

IS\_UART\_LIN\_WORD\_LENGTH

IS\_UART\_STOPBITS

IS\_UART\_PARITY

IS\_UART\_HARDWARE\_FLOW\_CONTROL

IS\_UART\_MODE

IS\_UART\_STATE

IS\_UART\_OVERSAMPLING

IS\_UART\_LIN\_OVERSAMPLING

IS\_UART\_LIN\_BREAK\_DETECT\_LENGTH

IS\_UART\_WAKEUPMETHOD

IS\_UART\_BAUDRATE

72 MHz) divided by the smallest  
oversampling used on the USART (i.e. 16)  
Retrun : TRUE or FALSE

IS\_UART\_ADDRESS

This parameter must be a number between  
Min\_Data = 0 and Max\_Data = 15 Return :  
TRUE or FALSE

UART\_IT\_MASK

**UART State**

UART\_STATE\_DISABLE

UART\_STATE\_ENABLE

**UART Number of Stop Bits**

UART\_STOPBITS\_1

UART\_STOPBITS\_2

**UART Wakeup Functions**

UART\_WAKEUPMETHOD\_IDLELINE

UART\_WAKEUPMETHOD\_ADDRESSMARK

**UART Word Length**

UART\_WORDLENGTH\_8B

UART\_WORDLENGTH\_9B

## 42 HAL USART Generic Driver

### 42.1 USART Firmware driver registers structures

#### 42.1.1 USART\_InitTypeDef

**USART\_InitTypeDef** is defined in the stm32f1xx\_hal\_usart.h

##### Data Fields

- **uint32\_t BaudRate**
- **uint32\_t WordLength**
- **uint32\_t StopBits**
- **uint32\_t Parity**
- **uint32\_t Mode**
- **uint32\_t CLKPolarity**
- **uint32\_t CLKPhase**
- **uint32\_t CLKLastBit**

##### Field Documentation

- **uint32\_t USART\_InitTypeDef::BaudRate** This member configures the Usart communication baud rate. The baud rate is computed using the following formula:
  - IntegerDivider = ((PCLKx) / (16 \* (husart->Init.BaudRate)))
  - FractionalDivider = ((IntegerDivider - ((uint32\_t) IntegerDivider)) \* 16) + 0.5
- **uint32\_t USART\_InitTypeDef::WordLength** Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [USART\\_Word\\_Length](#)
- **uint32\_t USART\_InitTypeDef::StopBits** Specifies the number of stop bits transmitted. This parameter can be a value of [USART\\_Stop\\_Bits](#)
- **uint32\_t USART\_InitTypeDef::Parity** Specifies the parity mode. This parameter can be a value of [USART\\_Parity](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- **uint32\_t USART\_InitTypeDef::Mode** Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [USART\\_Mode](#)
- **uint32\_t USART\_InitTypeDef::CLKPolarity** Specifies the steady state of the serial clock. This parameter can be a value of [USART\\_Clock\\_Polarity](#)
- **uint32\_t USART\_InitTypeDef::CLKPhase** Specifies the clock transition on which the bit capture is made. This parameter can be a value of [USART\\_Clock\\_Phase](#)
- **uint32\_t USART\_InitTypeDef::CLKLastBit** Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of [USART\\_Last\\_Bit](#)

#### 42.1.2 USART\_HandleTypeDef

**USART\_HandleTypeDef** is defined in the stm32f1xx\_hal\_usart.h

##### Data Fields

- **USART\_TypeDef \* Instance**

- **USART\_InitTypeDef Init**
- **uint8\_t \*pTxBuffPtr**
- **uint16\_t TxXferSize**
- **\_\_IO uint16\_t TxXferCount**
- **uint8\_t \*pRxBuffPtr**
- **uint16\_t RxXferSize**
- **\_\_IO uint16\_t RxXferCount**
- **DMA\_HandleTypeDef \*hdmatx**
- **DMA\_HandleTypeDef \*hdmarx**
- **HAL\_LockTypeDef Lock**
- **\_\_IO HAL\_USART\_StateTypeDef State**
- **\_\_IO uint32\_t ErrorCode**

#### Field Documentation

- **USART\_TypeDef\* USART\_HandleTypeDef::Instance** USART registers base address
- **USART\_InitTypeDef USART\_HandleTypeDef::Init** Usart communication parameters
- **uint8\_t\* USART\_HandleTypeDef::pTxBuffPtr** Pointer to Usart Tx transfer Buffer
- **uint16\_t USART\_HandleTypeDef::TxXferSize** Usart Tx Transfer size
- **\_\_IO uint16\_t USART\_HandleTypeDef::TxXferCount** Usart Tx Transfer Counter
- **uint8\_t\* USART\_HandleTypeDef::pRxBuffPtr** Pointer to Usart Rx transfer Buffer
- **uint16\_t USART\_HandleTypeDef::RxXferSize** Usart Rx Transfer size
- **\_\_IO uint16\_t USART\_HandleTypeDef::RxXferCount** Usart Rx Transfer Counter
- **DMA\_HandleTypeDef\* USART\_HandleTypeDef::hdmatx** Usart Tx DMA Handle parameters
- **DMA\_HandleTypeDef\* USART\_HandleTypeDef::hdmarx** Usart Rx DMA Handle parameters
- **HAL\_LockTypeDef USART\_HandleTypeDef::Lock** Locking object
- **\_\_IO HAL\_USART\_StateTypeDef USART\_HandleTypeDef::State** Usart communication state
- **\_\_IO uint32\_t USART\_HandleTypeDef::ErrorCode** USART Error code

## 42.2 USART Firmware driver API description

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

### 42.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART\_HandleTypeDef handle structure.
2. Initialize the USART low level resources by implementing the HAL\_USART\_MspInit() API:
  - a. Enable the USARTx interface clock.
  - b. USART pins configuration:
    - Enable the clock for the USART GPIOs.
    - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).

- c. NVIC configuration if you need to use interrupt process (HAL\_USART\_Transmit\_IT(), HAL\_USART\_Receive\_IT() and HAL\_USART\_TransmitReceive\_IT() APIs):
  - Configure the USARTx interrupt priority.
  - Enable the NVIC USART IRQ handle.
- d. DMA Configuration if you need to use DMA process (HAL\_USART\_Transmit\_DMA(), HAL\_USART\_Receive\_DMA() and HAL\_USART\_TransmitReceive\_DMA() APIs):
  - Declare a DMA handle structure for the Tx/Rx channel.
  - Enable the DMAx interface clock.
  - Configure the declared DMA handle structure with the required Tx/Rx parameters.
  - Configure the DMA Tx/Rx channel.
  - Associate the initialized DMA handle to the USART DMA Tx/Rx handle.
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
  - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the husart Init structure.
4. Initialize the USART registers by calling the HAL\_USART\_Init() API:
  - These APIs configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL\_USART\_MspInit(&husart) API. The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros \_\_HAL\_USART\_ENABLE\_IT() and \_\_HAL\_USART\_DISABLE\_IT() inside the transmit and receive process.
5. Three operation modes are available within this driver :

### Polling mode IO operation

- Send an amount of data in blocking mode using HAL\_USART\_Transmit()
- Receive an amount of data in blocking mode using HAL\_USART\_Receive()

### Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL\_USART\_Transmit\_IT()
- At transmission end of transfer HAL\_USART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_USART\_Receive\_IT()
- At reception end of transfer HAL\_USART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_RxCpltCallback
- In case of transfer Error, HAL\_USART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_USART\_ErrorCallback

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL\_USART\_Transmit\_DMA()
- At transmission end of half transfer HAL\_USART\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxHalfCpltCallback
- At transmission end of transfer HAL\_USART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_USART\_Receive\_DMA()
- At reception end of half transfer HAL\_USART\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_RxHalfCpltCallback
- At reception end of transfer HAL\_USART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_RxCpltCallback
- In case of transfer Error, HAL\_USART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_USART\_ErrorCallback
- Pause the DMA Transfer using HAL\_USART\_DMAPause()
- Resume the DMA Transfer using HAL\_USART\_DMAResume()
- Stop the DMA Transfer using HAL\_USART\_DMAStop()

### USART HAL driver macros list

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

- \_\_HAL\_USART\_ENABLE: Enable the USART peripheral
- \_\_HAL\_USART\_DISABLE: Disable the USART peripheral
- \_\_HAL\_USART\_GET\_FLAG : Check whether the specified USART flag is set or not
- \_\_HAL\_USART\_CLEAR\_FLAG : Clear the specified USART pending flag
- \_\_HAL\_USART\_ENABLE\_IT: Enable the specified USART interrupt
- \_\_HAL\_USART\_DISABLE\_IT: Disable the specified USART interrupt
- \_\_HAL\_USART\_GET\_IT\_SOURCE: Check whether the specified USART interrupt has occurred or not



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

## 42.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in 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), the possible USART frame formats are as listed in [Table 23: "USART frame formats"](#).
  - USART polarity



- USART phase
- USART LastBit
- Receiver/transmitter modes

**Table 23: USART frame formats**

M bit	PCE bit	USART frame
0	0	SB   8 bit data   STB
0	1	SB   7 bit data   PB   STB
1	0	SB   9 bit data   STB
1	1	SB   8 bit data   PB   STB

The HAL\_USART\_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manuals (RM0008 for STM32F10Xxx MCUs and RM0041 for STM32F100xx MCUs)).

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

### 42.2.3 IO operation functions

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

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

- There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA. These APIs return the HAL status. The end of the data processing will be indicated through the dedicated 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.
- Blocking mode APIs are :
  - HAL\_USART\_Transmit() in simplex mode
  - HAL\_USART\_Receive() in full duplex receive only
  - HAL\_USART\_TransmitReceive() in full duplex mode
- Non Blocking mode APIs with Interrupt are :
  - HAL\_USART\_Transmit\_IT() in simplex mode
  - HAL\_USART\_Receive\_IT() in full duplex receive only
  - HAL\_USART\_TransmitReceive\_IT() in full duplex mode
  - HAL\_USART\_IRQHandler()
- Non Blocking mode functions with DMA are :
  - HAL\_USART\_Transmit\_DMA() in simplex mode
  - HAL\_USART\_Receive\_DMA() in full duplex receive only

- HAL\_USART\_TransmitReceive\_DMA() in full duplex mode
- HAL\_USART\_DMAPause()
- HAL\_USART\_DMAResume()
- HAL\_USART\_DMAStop()
- 5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_USART\_TxCpltCallback()
  - HAL\_USART\_TxCpltCallback()
  - HAL\_USART\_RxCpltCallback()
  - HAL\_USART\_RxCpltCallback()
  - HAL\_USART\_ErrorCallback()
  - HAL\_USART\_TxRxCpltCallback()
- [HAL\\_USART\\_Transmit\(\)](#)
- [HAL\\_USART\\_Receive\(\)](#)
- [HAL\\_USART\\_TransmitReceive\(\)](#)
- [HAL\\_USART\\_Transmit\\_IT\(\)](#)
- [HAL\\_USART\\_Receive\\_IT\(\)](#)
- [HAL\\_USART\\_TransmitReceive\\_IT\(\)](#)
- [HAL\\_USART\\_Transmit\\_DMA\(\)](#)
- [HAL\\_USART\\_Receive\\_DMA\(\)](#)
- [HAL\\_USART\\_TransmitReceive\\_DMA\(\)](#)
- [HAL\\_USART\\_DMAPause\(\)](#)
- [HAL\\_USART\\_DMAResume\(\)](#)
- [HAL\\_USART\\_DMAStop\(\)](#)
- [HAL\\_USART\\_IRQHandler\(\)](#)
- [HAL\\_USART\\_TxCpltCallback\(\)](#)
- [HAL\\_USART\\_TxCpltCallback\(\)](#)
- [HAL\\_USART\\_RxCpltCallback\(\)](#)
- [HAL\\_USART\\_RxCpltCallback\(\)](#)
- [HAL\\_USART\\_TxRxCpltCallback\(\)](#)
- [HAL\\_USART\\_ErrorCallback\(\)](#)

#### 42.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of USART communication process, return Peripheral Errors occurred during communication process

- HAL\_USART\_GetState() API can be helpful to check in run-time the state of the USART peripheral.
- HAL\_USART\_GetError() check in run-time errors that could be occurred during communication.
- [HAL\\_USART\\_GetState\(\)](#)
- [HAL\\_USART\\_GetError\(\)](#)

#### 42.2.5 HAL\_USART\_Init

Function Name	<b>HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)</b>
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"> <li>• <b>husart</b>: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified</li> </ul>

---

 USART module.

Return values

- HAL status

## 42.2.6 HAL\_USART\_DeInit

Function Name **HAL\_StatusTypeDef HAL\_USART\_DeInit (USART\_HandleTypeDef \* husart)**

Function Description DeInitializes the USART peripheral.

Parameters

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- HAL status

## 42.2.7 HAL\_USART\_MspInit

Function Name **void HAL\_USART\_MspInit (USART\_HandleTypeDef \* husart)**

Function Description USART MSP Init.

Parameters

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- None

## 42.2.8 HAL\_USART\_MspDeInit

Function Name **void HAL\_USART\_MspDeInit (USART\_HandleTypeDef \* husart)**

Function Description USART MSP DeInit.

Parameters

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- None

## 42.2.9 HAL\_USART\_Transmit

Function Name **HAL\_StatusTypeDef HAL\_USART\_Transmit (USART\_HandleTypeDef \* husart, uint8\_t \* pTxData, uint16\_t Size, uint32\_t Timeout)**

Function Description Simplex Send an amount of data in blocking mode.

Parameters

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pTxData**: Pointer to data buffer

- **Size:** Amount of data to be sent
  - **Timeout:** Timeout duration
- Return values
- HAL status

#### 42.2.10 HAL\_USART\_Receive

- Function Name **HAL\_StatusTypeDef HAL\_USART\_Receive (USART\_HandleTypeDef \* husart, uint8\_t \* pRxData, uint16\_t Size, uint32\_t Timeout)**
- Function Description Full-Duplex Receive an amount of data in blocking mode.
- Parameters
- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
  - **pRxData:** Pointer to data buffer
  - **Size:** Amount of data to be received
  - **Timeout:** Timeout duration
- Return values
- HAL status

#### 42.2.11 HAL\_USART\_TransmitReceive

- Function Name **HAL\_StatusTypeDef HAL\_USART\_TransmitReceive (USART\_HandleTypeDef \* husart, uint8\_t \* pTxData, uint8\_t \* pRxData, uint16\_t Size, uint32\_t Timeout)**
- Function Description Full-Duplex Send receive an amount of data in full-duplex mode (blocking mode).
- Parameters
- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
  - **pTxData:** Pointer to data transmitted buffer
  - **pRxData:** Pointer to data received buffer
  - **Size:** Amount of data to be sent
  - **Timeout:** Timeout duration
- Return values
- HAL status

#### 42.2.12 HAL\_USART\_Transmit\_IT

- Function Name **HAL\_StatusTypeDef HAL\_USART\_Transmit\_IT (USART\_HandleTypeDef \* husart, uint8\_t \* pTxData, uint16\_t Size)**
- Function Description Simplex Send an amount of data in non-blocking mode.
- Parameters
- **husart:** Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
  - **pTxData:** Pointer to data buffer

	<ul style="list-style-type: none"> <li>• <b>Size:</b> Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The USART errors are not managed to avoid the overrun error.</li> </ul>

### 42.2.13 HAL\_USART\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive_IT</b> (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)
Function Description	Simplex Receive an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> <li>• <b>pRxData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 42.2.14 HAL\_USART\_TransmitReceive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive_IT</b> (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Full-Duplex Send receive an amount of data in full-duplex mode (non-blocking).
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> <li>• <b>pTxData:</b> Pointer to data transmitted buffer</li> <li>• <b>pRxData:</b> Pointer to data received buffer</li> <li>• <b>Size:</b> Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 42.2.15 HAL\_USART\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit_DMA</b> (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)
Function Description	Simplex Send an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart:</b> Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> <li>• <b>pTxData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> </ul>

Return values

- HAL status

#### 42.2.16 HAL\_USART\_Receive\_DMA

**Function Name** **HAL\_StatusTypeDef HAL\_USART\_Receive\_DMA (USART\_HandleTypeDef \* husart, uint8\_t \* pRxData, uint16\_t Size)**

**Function Description** Full-Duplex Receive an amount of data in non-blocking mode.

**Parameters**

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pRxData**: Pointer to data buffer
- **Size**: Amount of data to be received

**Return values**

- HAL status

**Notes**

- The USART DMA transmit channel must be configured in order to generate the clock for the slave.
- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

#### 42.2.17 HAL\_USART\_TransmitReceive\_DMA

**Function Name** **HAL\_StatusTypeDef HAL\_USART\_TransmitReceive\_DMA (USART\_HandleTypeDef \* husart, uint8\_t \* pTxData, uint8\_t \* pRxData, uint16\_t Size)**

**Function Description** Full-Duplex Transmit Receive an amount of data in non-blocking mode.

**Parameters**

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pTxData**: Pointer to data transmitted buffer
- **pRxData**: Pointer to data received buffer
- **Size**: Amount of data to be received

**Return values**

- HAL status

**Notes**

- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

#### 42.2.18 HAL\_USART\_DMAPause

**Function Name** **HAL\_StatusTypeDef HAL\_USART\_DMAPause (USART\_HandleTypeDef \* husart)**

**Function Description** Pauses the DMA Transfer.

**Parameters**

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- HAL status

#### 42.2.19 HAL\_USART\_DMAResume

Function Name **HAL\_StatusTypeDef HAL\_USART\_DMAResume (USART\_HandleTypeDef \* husart)**

Function Description Resumes the DMA Transfer.

Parameters

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- HAL status

#### 42.2.20 HAL\_USART\_DMAStop

Function Name **HAL\_StatusTypeDef HAL\_USART\_DMAStop (USART\_HandleTypeDef \* husart)**

Function Description Stops the DMA Transfer.

Parameters

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- HAL status

#### 42.2.21 HAL\_USART\_IRQHandler

Function Name **void HAL\_USART\_IRQHandler (USART\_HandleTypeDef \* husart)**

Function Description This function handles USART interrupt request.

Parameters

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- None

#### 42.2.22 HAL\_USART\_TxCpltCallback

Function Name **void HAL\_USART\_TxCpltCallback (USART\_HandleTypeDef \* husart)**

Function Description Tx Transfer completed callbacks.

Parameters

- **husart**: Pointer to a USART\_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- None

### 42.2.23 HAL\_USART\_TxHalfCpltCallback

Function Name	<b>void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)</b>
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b>: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 42.2.24 HAL\_USART\_RxCpltCallback

Function Name	<b>void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b>: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 42.2.25 HAL\_USART\_RxHalfCpltCallback

Function Name	<b>void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)</b>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b>: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 42.2.26 HAL\_USART\_TxRxCpltCallback

Function Name	<b>void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)</b>
Function Description	Tx/Rx Transfers completed callback for the non-blocking process.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b>: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 42.2.27 HAL\_USART\_ErrorCallback



Function Name	<b>void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)</b>
Function Description	USART error callbacks.
Parameters	<ul style="list-style-type: none"> <li><b>husart</b>: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 42.2.28 HAL\_USART\_GetState

Function Name	<b>HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)</b>
Function Description	Returns the USART state.
Parameters	<ul style="list-style-type: none"> <li><b>husart</b>: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

#### 42.2.29 HAL\_USART\_GetError

Function Name	<b>uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)</b>
Function Description	Return the USART error code.
Parameters	<ul style="list-style-type: none"> <li><b>husart</b>: : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>USART Error Code</li> </ul>

### 42.3 USART Firmware driver defines

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

#### 42.3.1 USART

USART

**USART Clock**

USART\_CLOCK\_DISABLE

USART\_CLOCK\_ENABLE

**USART Clock Phase**

USART\_PHASE\_1EDGE

USART\_PHASE\_2EDGE

**USART Clock Polarity**

USART\_POLARITY\_LOW

USART\_POLARITY\_HIGH

### **USART Error Codes**

HAL_USART_ERROR_NONE	No error
HAL_USART_ERROR_PE	Parity error
HAL_USART_ERROR_NE	Noise error
HAL_USART_ERROR_FE	frame error
HAL_USART_ERROR_ORE	Overrun error
HAL_USART_ERROR_DMA	DMA transfer error

### **USART Exported Macros**

`__HAL_USART_RESET_HANDLE_STATE` **Description:**

- Reset USART handle state.

#### **Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

#### **Return value:**

- None:

`__HAL_USART_GET_FLAG`

#### **Description:**

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

#### **Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - USART\_FLAG\_TXE: Transmit data register empty flag
  - USART\_FLAG\_TC: Transmission Complete flag
  - USART\_FLAG\_RXNE: Receive data register not empty flag
  - USART\_FLAG\_IDLE: Idle Line detection flag
  - USART\_FLAG\_ORE: OverRun Error flag
  - USART\_FLAG\_NE: Noise Error flag
  - USART\_FLAG\_FE: Framing Error flag
  - USART\_FLAG\_PE: Parity Error

flag

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

**Description:**

- Clear the specified USART pending flags.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be any combination of the following values:
  - USART\_FLAG\_TC: Transmission Complete flag.
  - USART\_FLAG\_RXNE: Receive data register not empty flag.

**Return value:**

- None:

**Description:**

- Clear the USART PE pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

**Description:**

- Clear the USART FE pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

**Description:**

- Clear the USART NE pending flag.

**Parameters:**

\_\_HAL\_USART\_CLEAR\_FLAG

\_\_HAL\_USART\_CLEAR\_PEFLAG

\_\_HAL\_USART\_CLEAR\_FEFLAG

\_\_HAL\_USART\_CLEAR\_NEFLAG

`__HAL_USART_CLEAR_OREFLAG`

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

**Description:**

- Clear the USART ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

`__HAL_USART_CLEAR_IDLEFLAG`**Description:**

- Clear the USART IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

`__HAL_USART_ENABLE_IT`**Description:**

- Enable the specified Usart interrupts.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__INTERRUPT__`: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - `USART_IT_TXE`: Transmit Data Register empty interrupt
  - `USART_IT_TC`: Transmission complete interrupt
  - `USART_IT_RXNE`: Receive Data register not empty interrupt
  - `USART_IT_IDLE`: Idle line detection interrupt
  - `USART_IT_PE`: Parity Error interrupt

- USART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

**Description:**

- Disable the specified Usart interrupts.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- **\_\_INTERRUPT\_\_**: specifies the USART interrupt source to disable. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_PE: Parity Error interrupt
  - USART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

**Description:**

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

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- **\_\_IT\_\_**: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt

`__HAL_USART_DISABLE_IT``__HAL_USART_GET_IT_SOURCE`

- USART\_IT\_IDLE: Idle line detection interrupt
- USART\_IT\_ERR: Error interrupt
- USART\_IT\_PE: Parity Error interrupt

**Return value:**

- The: new state of \_\_IT\_\_ (TRUE or FALSE).

**Description:**

- Enable USART.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

**Description:**

- Disable USART.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None:

\_\_HAL\_USART\_ENABLE

\_\_HAL\_USART\_DISABLE

**USART Flags**

USART\_FLAG\_CTS

USART\_FLAG\_LBD

USART\_FLAG\_TXE

USART\_FLAG\_TC

USART\_FLAG\_RXNE

USART\_FLAG\_IDLE

USART\_FLAG\_ORE

USART\_FLAG\_NE

USART\_FLAG\_FE

USART\_FLAG\_PE

**USART Interrupts Definition**

USART\_IT\_PE

USART\_IT\_TXE

USART\_IT\_TC

USART\_IT\_RXNE

USART\_IT\_IDLE

USART\_IT\_LBD

USART\_IT\_CTS

USART\_IT\_ERR

**USART Last Bit**

USART\_LASTBIT\_DISABLE

USART\_LASTBIT\_ENABLE

**USART Mode**

USART\_MODE\_RX

USART\_MODE\_TX

USART\_MODE\_TX\_RX

**USART NACK State**

USART\_NACK\_ENABLE

USART\_NACK\_DISABLE

**USART Parity**

USART\_PARITY\_NONE

USART\_PARITY\_EVEN

USART\_PARITY\_ODD

**USART Private Constants**

DUMMY\_DATA

**USART Private Macros**

USART\_CR1\_REG\_INDEX

USART\_CR2\_REG\_INDEX

USART\_CR3\_REG\_INDEX

USART\_DIV

USART\_DIVMANT

USART\_DIVFRAQ

USART\_BRR

IS\_USART\_BAUDRATE      72 MHz) divided by the smallest oversampling used on the USART (i.e. 16) return : TRUE or FALSE

IS\_USART\_WORD\_LENGTH

IS\_USART\_STOPBITS

IS\_USART\_PARITY

IS\_USART\_MODE

IS\_USART\_CLOCK

IS\_USART\_POLARITY

IS\_USART\_PHASE

IS\_USART\_LASTBIT

IS\_USART\_NACK\_STATE

USART\_IT\_MASK

***USART Number of Stop Bits***

USART\_STOPBITS\_1

USART\_STOPBITS\_0\_5

USART\_STOPBITS\_2

USART\_STOPBITS\_1\_5

***USART Word Length***

USART\_WORDLENGTH\_8B

USART\_WORDLENGTH\_9B



## 43 HAL WWDG Generic Driver

### 43.1 WWDG Firmware driver registers structures

#### 43.1.1 WWDG\_InitTypeDef

**WWDG\_InitTypeDef** is defined in the stm32f1xx\_hal\_wwdg.h

##### Data Fields

- **uint32\_t Prescaler**
- **uint32\_t Window**
- **uint32\_t Counter**

##### Field Documentation

- **uint32\_t WWDG\_InitTypeDef::Prescaler** Specifies the prescaler value of the WWDG. This parameter can be a value of [WWDG\\_Prescaler](#)
- **uint32\_t WWDG\_InitTypeDef::Window** Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number 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

#### 43.1.2 WWDG\_HandleTypeDef

**WWDG\_HandleTypeDef** is defined in the stm32f1xx\_hal\_wwdg.h

##### Data Fields

- **WWDG\_TypeDef \* Instance**
- **WWDG\_InitTypeDef Init**
- **HAL\_LockTypeDef Lock**
- **\_\_IO HAL\_WWDG\_StateTypeDef State**

##### Field Documentation

- **WWDG\_TypeDef\* WWDG\_HandleTypeDef::Instance** Register base address
- **WWDG\_InitTypeDef WWDG\_HandleTypeDef::Init** WWDG required parameters
- **HAL\_LockTypeDef WWDG\_HandleTypeDef::Lock** WWDG locking object
- **\_\_IO HAL\_WWDG\_StateTypeDef WWDG\_HandleTypeDef::State** WWDG communication state

### 43.2 WWDG Firmware driver API description

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

#### 43.2.1 WWDG specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the Counter (T[6;0] 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.
- WWDGRST 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.
- $\text{WWDG clock (Hz)} = \text{PCLK} / (4096 * \text{Prescaler})$
- $\text{WWDG timeout (mS)} = 1000 * (T[5;0] + 1) / \text{WWDG clock}$  where T[5;0] are the lowest 6 bits of Counter.
- WWDG Counter refresh is allowed between the following limits :
  - min time (mS) =  $1000 * (\text{Counter} - \text{Window}) / \text{WWDG clock}$
  - max time (mS) =  $1000 * (\text{Counter} - 0x40) / \text{WWDG clock}$
- Min-max timeout value @48 MHz(PCLK): ~85,3us / ~5,46 ms

### 43.2.2 How to use this driver

- Enable WWDG APB1 clock using `__HAL_RCC_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()`. At EWI `HAL_WWDG_WakeupCallback` is executed and user can add his own code by customization of function pointer `HAL_WWDG_WakeupCallback`. 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

### 43.2.3 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\(\)](#)
- [HAL\\_WWDG\\_WakeupCallback\(\)](#)

#### 43.2.4 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\(\)](#)

#### 43.2.5 Peripheral State functions

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

- [HAL\\_WWDG\\_GetState\(\)](#)

#### 43.2.6 HAL\_WWDG\_Init

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Init</b> (WWDG_HandleTypeDef * hwwdg)
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"> <li>• <b>hwwdg</b>: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 43.2.7 HAL\_WWDG\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_DeInit</b> (WWDG_HandleTypeDef * hwwdg)
Function Description	DeInitializes the WWDG peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwwdg</b>: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified</li> </ul>

---

WWDG module.

Return values

- HAL status

### 43.2.8 HAL\_WWDG\_Msplnit

Function Name **void HAL\_WWDG\_Msplnit (WWDG\_HandleTypeDef \* hwwdg)**

Function Description Initializes the WWDG MSP.

Parameters

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

Return values

- None

### 43.2.9 HAL\_WWDG\_MspDeInit

Function Name **void HAL\_WWDG\_MspDeInit (WWDG\_HandleTypeDef \* hwwdg)**

Function Description DeInitializes the WWDG MSP.

Parameters

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

Return values

- None

### 43.2.10 HAL\_WWDG\_WakeupCallback

Function Name **void HAL\_WWDG\_WakeupCallback (WWDG\_HandleTypeDef \* hwwdg)**

Function Description Early Wakeup WWDG callback.

Parameters

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

Return values

- None

### 43.2.11 HAL\_WWDG\_Start

Function Name **HAL\_StatusTypeDef HAL\_WWDG\_Start (WWDG\_HandleTypeDef \* hwwdg)**

Function Description Starts the WWDG.

Parameters

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

Return values

- HAL status

**43.2.12 HAL\_WWDG\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Start_IT (WWDG_HandleTypeDef * hwwdg)</b>
Function Description	Starts the WWDG with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li><b>hwwdg</b>: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**43.2.13 HAL\_WWDG\_Refresh**

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg, uint32_t Counter)</b>
Function Description	Refreshes the WWDG.
Parameters	<ul style="list-style-type: none"> <li><b>hwwdg</b>: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> <li><b>Counter</b>: value of counter to put in WWDG counter</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**43.2.14 HAL\_WWDG\_IRQHandler**

Function Name	<b>void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)</b>
Function Description	Handles WWDG interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hwwdg</b>: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>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 when calling HAL_WWDG_Start_IT function. 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.</li> </ul>

**43.2.15 HAL\_WWDG\_WakeupCallback**

Function Name	<b>void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwwdg)</b>
---------------	--

Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"> <li><b>hwwdg</b>: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 43.2.16 HAL\_WWDG\_GetState

Function Name	<b>HAL_WWDG_StateTypeDef HAL_WWDG_GetState(WWDG_HandleTypeDef * hwwdg)</b>
Function Description	Returns the WWDG state.
Parameters	<ul style="list-style-type: none"> <li><b>hwwdg</b>: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

## 43.3 WWDG Firmware driver defines

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

### 43.3.1 WWDG

WWDG

#### **WWDG Exported Macros**

<b>__HAL_WWDG_RESET_HANDLE_STATE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset WWDG handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__</b>: WWDG handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None:</li> </ul>
<b>__HAL_WWDG_ENABLE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Enables the WWDG peripheral.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__</b>: WWDG handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None:</li> </ul>
<b>__HAL_WWDG_DISABLE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Disables the WWDG peripheral.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__</b>: WWDG handle</li> </ul> <b>Return value:</b>

**\_\_HAL\_WWDG\_ENABLE\_IT**

- None:

**Description:**

- Enables the WWDG early wakeup interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: WWDG handle
- **\_\_INTERRUPT\_\_**: specifies the interrupt to enable. This parameter can be one of the following values:
  - **WWDG\_IT\_EWI**: Early wakeup interrupt

**Return value:**

- None:

**\_\_HAL\_WWDG\_DISABLE\_IT****Description:**

- Disables the WWDG early wakeup interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: WWDG handle
- **\_\_INTERRUPT\_\_**: specifies the interrupt to disable. This parameter can be one of the following values:
  - **WWDG\_IT\_EWI**: Early wakeup interrupt

**Return value:**

- None:

**\_\_HAL\_WWDG\_GET\_IT****Description:**

- Gets the selected WWDG's it status.

**Parameters:**

- **\_\_HANDLE\_\_**: WWDG handle
- **\_\_INTERRUPT\_\_**: specifies the it to check. This parameter can be one of the following values:
  - **WWDG\_FLAG\_EWIF**: Early wakeup interrupt IT

**Return value:**

- The: new state of **WWDG\_FLAG** (SET or RESET).

**\_\_HAL\_WWDG\_CLEAR\_IT****Description:**

- Clear the WWDG's interrupt pending bits bits to clear the selected interrupt pending bits.

**Parameters:**

- **\_\_HANDLE\_\_**: WWDG handle

`__HAL_WWDG_GET_FLAG`

- `__INTERRUPT__`: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

**Description:**

- Gets the selected WWDG's flag status.

**Parameters:**

- `__HANDLE__`: WWDG handle
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

**Return value:**

- The: new state of `WWDG_FLAG` (SET or RESET).

`__HAL_WWDG_CLEAR_FLAG`**Description:**

- Clears the WWDG's pending flags.

**Parameters:**

- `__HANDLE__`: WWDG handle
- `__FLAG__`: specifies the flag to clear. This parameter can be one of the following values:
  - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

**Return value:**

- None:

`__HAL_WWDG_GET_IT_SOURCE`**Description:**

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

**Parameters:**

- `__HANDLE__`: WWDG Handle.
- `__INTERRUPT__`: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
  - `WWDG_IT_EWI`: Early Wakeup Interrupt

**Return value:**

- state: of `__INTERRUPT__` (TRUE or FALSE).

**WWDG Flag definition**



WWDG\_FLAG\_EWIF    Early wakeup interrupt flag

**WWDG Interrupt definition**

WWDG\_IT\_EWI        Early wakeup interrupt

**WWDG Prescaler**

WWDG\_PRESCALER\_1    WWDG counter clock = (PCLK1/4096)/1

WWDG\_PRESCALER\_2    WWDG counter clock = (PCLK1/4096)/2

WWDG\_PRESCALER\_4    WWDG counter clock = (PCLK1/4096)/4

WWDG\_PRESCALER\_8    WWDG counter clock = (PCLK1/4096)/8

**WWDG Private Macros**

IS\_WWDG\_PRESCALER

IS\_WWDG\_WINDOW

IS\_WWDG\_COUNTER

## 44 FAQs

### General subjects

#### Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not required in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, ...). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL\_UART\_Init() then HAL\_UART\_Transmit() or HAL\_UART\_Receive().

#### Which STM32F1 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32F1 devices. To ensure compatibility between all devices and portability with others series and lines, the API is split into the generic and the extension APIs . For more details, please refer to [Section 4.4: "Devices supported by HAL drivers"](#).

#### What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

### Architecture

#### How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32f1xx\_hal\_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration...)

A template is provided in the HAL drivers folders (stm32f1xx\_hal\_conf\_template.c).

### Which header files should I include in my application to use the HAL drivers?

Only stm32f1xx\_hal.h file has to be included.

### What is the difference between stm32f1xx\_hal\_ppp.c/h and stm32f1xx\_hal\_ppp\_ex.c/h?

The HAL driver architecture supports common features across STM32 series/lines. To support specific features, the drivers are split into two groups.

- The generic APIs (stm32f1xx\_hal\_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32f1xx\_hal\_ppp\_ex.c): It includes the specific APIs for specific device part number or family.

## Initialization and I/O operation functions

### How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system\_stm32f1xx.c) but in the main user application by calling the two main functions, HAL\_RCC\_OscConfig() and HAL\_RCC\_ClockConfig(). It can be modified in any user application section.

### What is the purpose of the *PPP\_HandleTypeDef \*pHandle* structure located in each driver in addition to the Initialization structure

*PPP\_HandleTypeDef \*pHandle* is the main structure implemented in the HAL drivers. It handles the peripheral configuration and registers, and embeds all the structures and variables required to follow the peripheral device flow (pointer to buffer, Error code, State,...)

However, this structure is not required to service peripherals such as GPIO, SYSTICK, PWR, and RCC.

### What is the purpose of HAL\_PPP\_MspInit() and HAL\_PPP\_MspDeInit() functions?

These function are called within HAL\_PPP\_Init() and HAL\_PPP\_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in stm32f1xx\_hal\_msp.c. A template is provided in the HAL driver folders (stm32f1xx\_hal\_msp\_template.c).

### When and how should I use callbacks functions (functions declared with the attribute *\_\_weak*)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

**Is it mandatory to use HAL\_Init() function at the beginning of the user application?**

It is mandatory to use HAL\_Init() function to enable the system configuration (Prefetch, Data instruction cache,...), configure the SysTick and the NVIC priority grouping and the hardware low level initialization.

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

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

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

The call HAL\_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL\_Delay().

**Why is the SysTick timer configured to have 1 ms?**

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

**Could HAL\_Delay() function block my application under certain conditions?**

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

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

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

1. Call HAL\_Init() function to initialize the system (data cache, NVIC priority,...).
2. Initialize the system clock by calling HAL\_RCC\_OscConfig() followed by HAL\_RCC\_ClockConfig().
3. Add HAL\_IncTick() function under SysTick\_Handler() ISR function to enable polling process when using HAL\_Delay() function
4. Start initializing your peripheral by calling HAL\_PPP\_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,...) by calling HAL\_PPP\_MspInit() in stm32f1xx\_hal\_msp.c
6. Start your process operation by calling IO operation functions.

**What is the purpose of HAL\_PPP\_IRQHandler() function and when should I use it?**

HAL\_PPP\_IRQHandler() is used to handle interrupt process. It is called under PPP\_IRQHandler() function in stm32f1xx\_it.c. In this case, the end-user has to implement only the callbacks functions (prefixed by \_\_weak) to perform the appropriate action when an interrupt is detected. Advanced users can implement their own code in PPP\_IRQHandler() without calling HAL\_PPP\_IRQHandler().

**Can I use directly the macros defined in stm32f1xx\_hal\_ppp.h ?**

Yes, you can: a set of macros is provided with the APIs. They allow accessing directly some specific features using peripheral flags.

**Where must PPP\_HandleTypedef structure peripheral handler be declared?**

PPP\_HandleTypedef structure peripheral handler must be declared as a global variable, so that all the structure fields are set to 0 by default. In this way, the peripheral handler default state are set to HAL\_PPP\_STATE\_RESET, which is the default state for each peripheral after a system reset.

## 45 Revision history

Table 24: Document revision history

Date	Revision	Changes
05-Feb-2015	1	Initial release.

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved

