

## Description of STM32F0xx 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 STM32CubeF0 for STM32F0 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 Acronyms and definitions.....</b>	<b>35</b>
<b>2 Overview of HAL drivers .....</b>	<b>37</b>
2.1 HAL and user-application files.....	37
2.1.1 HAL driver files .....	37
2.1.2 User-application files .....	38
2.2 HAL data structures .....	40
2.2.1 Peripheral handle structures .....	40
2.2.2 Initialization and configuration structure .....	41
2.2.3 Specific process structures .....	42
2.3 API classification .....	42
2.4 Devices supported by HAL drivers .....	43
2.5 HAL drivers rules.....	45
2.5.1 HAL API naming rules .....	45
2.5.2 HAL general naming rules.....	46
2.5.3 HAL interrupt handler and callback functions.....	47
2.6 HAL generic APIs.....	48
2.7 HAL extension APIs .....	49
2.7.1 HAL extension model overview .....	49
2.7.2 HAL extension model cases.....	49
2.8 File inclusion model.....	51
2.9 HAL common resources.....	52
2.10 HAL configuration.....	53
2.11 HAL system peripheral handling .....	54
2.11.1 Clock.....	54
2.11.2 GPIOs.....	55
2.11.3 Cortex NVIC and SysTick timer.....	56
2.11.4 PWR .....	57
2.11.5 EXTI.....	57
2.11.6 DMA.....	58
2.12 How to use HAL drivers .....	60
2.12.1 HAL usage models .....	60
2.12.2 HAL initialization .....	60
2.12.3 HAL IO operation process .....	62
2.12.4 Timeout and error management.....	65
<b>3 HAL System Driver .....</b>	<b>70</b>

3.1	HAL Firmware driver API description .....	70
3.1.1	How to use this driver .....	70
3.1.2	Initialization and de-initialization functions .....	70
3.1.3	HAL Control functions.....	70
3.1.4	HAL_Init.....	71
3.1.5	HAL_DelInit .....	71
3.1.6	HAL_MspInit.....	72
3.1.7	HAL_MspDeInit .....	72
3.1.8	HAL_InitTick .....	72
3.1.9	HAL_IncTick .....	73
3.1.10	HAL_GetTick .....	73
3.1.11	HAL_SuspendTick.....	73
3.1.12	HAL_ResumeTick.....	74
3.1.13	HAL_Delay .....	74
3.1.14	HAL_GetHalVersion .....	75
3.1.15	HAL_GetREVID .....	75
3.1.16	HAL_GetDEVID.....	75
3.1.17	HAL_EnableDBGStopMode .....	75
3.1.18	HAL_DisableDBGStopMode .....	76
3.1.19	HAL_EnableDBGStandbyMode .....	76
3.1.20	HAL_DisableDBGStandbyMode .....	76
3.2	HAL Firmware driver defines.....	77
3.2.1	HAL.....	77
4	<b>HAL ADC Generic Driver.....</b>	<b>81</b>
4.1	ADC Firmware driver registers structures .....	81
4.1.1	ADC_InitTypeDef.....	81
4.1.2	ADC_ChannelConfTypeDef .....	82
4.1.3	ADC_AnalogWDGConfTypeDef.....	83
4.1.4	ADC_HandleTypeDef .....	84
4.2	ADC Firmware driver API description.....	84
4.2.1	ADC specific features .....	84
4.2.2	How to use this driver .....	85
4.2.3	Initialization and de-initialization functions .....	85
4.2.4	IO operation functions .....	86
4.2.5	Peripheral Control functions .....	86
4.2.6	Peripheral State and Errors functions .....	86
4.2.7	HAL_ADC_Init .....	87
4.2.8	HAL_ADC_DelInit.....	87

4.2.9	HAL_ADC_MspInit .....	88
4.2.10	HAL_ADC_MspDeInit.....	88
4.2.11	HAL_ADC_Start .....	88
4.2.12	HAL_ADC_Stop.....	89
4.2.13	HAL_ADC_PollForConversion .....	89
4.2.14	HAL_ADC_PollForEvent .....	89
4.2.15	HAL_ADC_Start_IT .....	90
4.2.16	HAL_ADC_Stop_IT .....	90
4.2.17	HAL_ADC_Start_DMA .....	90
4.2.18	HAL_ADC_Stop_DMA.....	91
4.2.19	HAL_ADC_GetValue .....	91
4.2.20	HAL_ADC_IRQHandler.....	92
4.2.21	HAL_ADC_ConvCpltCallback .....	92
4.2.22	HAL_ADC_ConvHalfCpltCallback.....	92
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 .....	95
4.3	ADC Firmware driver defines .....	95
4.3.1	ADC .....	95
<b>5</b>	<b>HAL ADC Extension Driver .....</b>	<b>106</b>
5.1	ADCEx Firmware driver API description .....	106
5.1.1	ADC specific features .....	106
5.1.2	How to use this driver .....	106
5.1.3	IO operation functions .....	107
5.1.4	HAL_ADCEx_Calibration_Start.....	107
5.2	ADCEx Firmware driver defines .....	107
5.2.1	ADCEx .....	107
<b>6</b>	<b>HAL CAN Generic Driver .....</b>	<b>109</b>
6.1	CAN Firmware driver registers structures .....	109
6.1.1	CAN_InitTypeDef.....	109
6.1.2	CAN_FilterConfTypeDef.....	109
6.1.3	CanTxMsgTypeDef.....	110
6.1.4	CanRxMsgTypeDef .....	111
6.1.5	CAN_HandleTypeDef .....	112
6.2	CAN Firmware driver API description.....	112

6.2.1	How to use this driver .....	112
6.2.2	Initialization and de-initialization functions .....	113
6.2.3	IO operation functions .....	113
6.2.4	Peripheral State and Error functions .....	114
6.2.5	HAL_CAN_Init .....	114
6.2.6	HAL_CAN_ConfigFilter.....	114
6.2.7	HAL_CAN_DelInit.....	115
6.2.8	HAL_CAN_MspInit .....	115
6.2.9	HAL_CAN_MspDelInit.....	115
6.2.10	HAL_CAN_Transmit.....	116
6.2.11	HAL_CAN_Transmit_IT.....	116
6.2.12	HAL_CAN_Receive.....	117
6.2.13	HAL_CAN_Receive_IT.....	117
6.2.14	HAL_CAN_Sleep.....	117
6.2.15	HAL_CAN_WakeUp .....	118
6.2.16	HAL_CAN_IRQHandler.....	118
6.2.17	HAL_CAN_TxCpltCallback.....	118
6.2.18	HAL_CAN_RxCpltCallback .....	119
6.2.19	HAL_CAN_ErrorCallback .....	119
6.2.20	HAL_CAN_GetState.....	119
6.2.21	HAL_CAN_GetError .....	120
6.3	CAN Firmware driver defines .....	120
6.3.1	CAN .....	120
<b>7</b>	<b>HAL CEC Generic Driver .....</b>	<b>129</b>
7.1	CEC Firmware driver registers structures .....	129
7.1.1	CEC_InitTypeDef.....	129
7.1.2	CEC_HandleTypeDef .....	130
7.2	CEC Firmware driver API description.....	130
7.2.1	How to use this driver .....	131
7.2.2	Initialization and Configuration functions.....	131
7.2.3	IO operation function .....	131
7.2.4	Peripheral Control function.....	132
7.2.5	HAL_CEC_Init .....	132
7.2.6	HAL_CEC_DelInit.....	132
7.2.7	HAL_CEC_MspInit .....	132
7.2.8	HAL_CEC_MspDelInit.....	133
7.2.9	HAL_CEC_Transmit .....	133
7.2.10	HAL_CEC_Receive .....	134

7.2.11	HAL_CEC_Transmit_IT.....	134
7.2.12	HAL_CEC_Receive_IT.....	134
7.2.13	HAL_CEC_IRQHandler.....	135
7.2.14	HAL_CEC_TxCpltCallback.....	135
7.2.15	HAL_CEC_RxCpltCallback .....	136
7.2.16	HAL_CEC_ErrorCallback .....	136
7.2.17	HAL_CEC_GetState.....	136
7.2.18	HAL_CEC_GetError .....	137
7.3	CEC Firmware driver defines .....	137
7.3.1	CEC .....	137
<b>8</b>	<b>HAL COMP Generic Driver.....</b>	<b>146</b>
8.1	COMP Firmware driver registers structures .....	146
8.1.1	COMP_InitTypeDef .....	146
8.1.2	COMP_HandleTypeDef .....	146
8.2	COMP Firmware driver API description .....	147
8.2.1	COMP Peripheral features .....	147
8.2.2	How to use this driver .....	148
8.2.3	Initialization and Configuration functions.....	148
8.2.4	IO operation functions .....	148
8.2.5	Peripheral Control functions .....	149
8.2.6	Peripheral State functions .....	149
8.2.7	HAL_COMP_Init .....	149
8.2.8	HAL_COMP_DeInit .....	149
8.2.9	HAL_COMP_MspInit .....	150
8.2.10	HAL_COMP_MspDeInit.....	150
8.2.11	HAL_COMP_Start .....	150
8.2.12	HAL_COMP_Stop .....	151
8.2.13	HAL_COMP_Start_IT .....	151
8.2.14	HAL_COMP_Stop_IT .....	151
8.2.15	HAL_COMP_IRQHandler.....	152
8.2.16	HAL_COMP_Lock .....	152
8.2.17	HAL_COMP_GetOutputLevel .....	152
8.2.18	HAL_COMP_TriggerCallback .....	153
8.2.19	HAL_COMP_GetState.....	153
8.3	COMP Firmware driver defines .....	153
8.3.1	COMP .....	153
<b>9</b>	<b>HAL CORTEX Generic Driver.....</b>	<b>159</b>
9.1	CORTEX Firmware driver API description .....	159

9.1.1	How to use this driver .....	159
9.1.2	Initialization and de-initialization functions .....	159
9.1.3	Peripheral Control functions .....	160
9.1.4	HAL_NVIC_SetPriority .....	160
9.1.5	HAL_NVIC_EnableIRQ .....	160
9.1.6	HAL_NVIC_DisableIRQ.....	161
9.1.7	HAL_NVIC_SystemReset.....	161
9.1.8	HAL_SYSTICK_Config.....	162
9.1.9	HAL_NVIC_GetPriority.....	162
9.1.10	HAL_NVIC_SetPendingIRQ.....	162
9.1.11	HAL_NVIC_GetPendingIRQ .....	163
9.1.12	HAL_NVIC_ClearPendingIRQ.....	163
9.1.13	HAL_SYSTICK_CLKSourceConfig .....	163
9.1.14	HAL_SYSTICK_IRQHandler .....	164
9.1.15	HAL_SYSTICK_Callback .....	164
9.2	CORTEX Firmware driver defines.....	164
9.2.1	CORTEX.....	164
<b>10</b>	<b>HAL CRC Generic Driver.....</b>	<b>166</b>
10.1	CRC Firmware driver registers structures .....	166
10.1.1	CRC_InitTypeDef .....	166
10.1.2	CRC_HandleTypeDef.....	167
10.2	CRC Firmware driver API description .....	167
10.2.1	How to use this driver .....	167
10.2.2	Initialization and Configuration functions.....	168
10.2.3	Peripheral Control functions .....	168
10.2.4	Peripheral State functions .....	168
10.2.5	HAL_CRC_Init .....	168
10.2.6	HAL_CRC_DeInit .....	169
10.2.7	HAL_CRC_MspInit .....	169
10.2.8	HAL_CRC_MspDeInit.....	169
10.2.9	HAL_CRC_Accumulate.....	170
10.2.10	HAL_CRC_Calculate.....	170
10.2.11	HAL_CRC_GetState.....	170
10.3	CRC Firmware driver defines .....	171
10.3.1	CRC.....	171
<b>11</b>	<b>HAL CRC Extension Driver .....</b>	<b>173</b>
11.1	CRCEEx Firmware driver API description .....	173

11.1.1	Product specific features .....	173
11.1.2	How to use this driver .....	173
11.1.3	Initialization and Configuration functions.....	173
11.1.4	HAL_CRCEx_Init.....	173
11.1.5	HAL_CRCEx_Input_Data_Reverse .....	173
11.1.6	HAL_CRCEx_Output_Data_Reverse.....	174
11.1.7	HAL_CRCEx_Polynomial_Set .....	174
11.2	CRCEx Firmware driver defines.....	175
11.2.1	CRCEx.....	175
<b>12</b>	<b>HAL DAC Generic Driver.....</b>	<b>177</b>
12.1	DAC Firmware driver registers structures .....	177
12.1.1	DAC_HandleTypeDef .....	177
12.1.2	DAC_ChannelConfTypeDef .....	177
12.2	DAC Firmware driver API description.....	177
12.2.1	DAC Peripheral features.....	178
12.2.2	How to use this driver .....	179
12.2.3	Initialization and de-initialization functions .....	180
12.2.4	IO operation functions .....	180
12.2.5	Peripheral Control functions .....	180
12.2.6	Peripheral State and Errors functions .....	180
12.2.7	HAL_DAC_Init .....	181
12.2.8	HAL_DAC_DelInit.....	181
12.2.9	HAL_DAC_MspInit .....	181
12.2.10	HAL_DAC_MspDelInit.....	182
12.2.11	HAL_DAC_Start .....	182
12.2.12	HAL_DAC_Stop.....	182
12.2.13	HAL_DAC_Start_DMA .....	183
12.2.14	HAL_DAC_Stop_DMA.....	183
12.2.15	HAL_DAC_ConvCpltCallbackCh1 .....	184
12.2.16	HAL_DAC_ConvHalfCpltCallbackCh1 .....	184
12.2.17	HAL_DAC_ErrorCallbackCh1 .....	185
12.2.18	HAL_DAC_DMAUnderrunCallbackCh1 .....	185
12.2.19	HAL_DAC_ConfigChannel .....	185
12.2.20	HAL_DAC_SetValue .....	186
12.2.21	HAL_DAC_GetValue .....	186
12.2.22	HAL_DAC_GetState .....	187
12.2.23	HAL_DAC_GetError .....	187
12.2.24	HAL_DAC_IRQHandler .....	187

12.3	DAC Firmware driver defines .....	188
12.3.1	DAC .....	188
<b>13</b>	<b>HAL DAC Extension Driver .....</b>	<b>192</b>
13.1	DACE Firmware driver API description .....	192
13.1.1	How to use this driver .....	192
13.1.2	Extended features functions .....	192
13.1.3	HAL_DAC_ConfigChannel .....	192
13.1.4	HAL_DAC_GetValue .....	193
13.1.5	HAL_DAC_Start .....	193
13.1.6	HAL_DAC_Start_DMA .....	194
13.1.7	HAL_DAC_IRQHandler .....	194
13.1.8	HAL_DACE_DualGetValue .....	195
13.1.9	HAL_DACE_TriangleWaveGenerate .....	195
13.1.10	HAL_DACE_NoiseWaveGenerate .....	196
13.1.11	HAL_DACE_DualSetValue .....	197
13.1.12	HAL_DACE_ConvCpltCallbackCh2 .....	197
13.1.13	HAL_DACE_ConvHalfCpltCallbackCh2 .....	198
13.1.14	HAL_DACE_ErrorCallbackCh2 .....	198
13.1.15	HAL_DACE_DMAUnderrunCallbackCh2 .....	198
13.1.16	DAC_DMAConvCpltCh2 .....	199
13.1.17	DAC_DMAMhalfConvCpltCh2 .....	199
13.1.18	DAC_DMAErrorCh2 .....	200
13.2	DACE Firmware driver defines .....	200
13.2.1	DACE .....	200
<b>14</b>	<b>HAL DMA Generic Driver .....</b>	<b>202</b>
14.1	DMA Firmware driver registers structures .....	202
14.1.1	DMA_InitTypeDef .....	202
14.1.2	__DMA_HandleTypeDef .....	202
14.2	DMA Firmware driver API description .....	203
14.2.1	How to use this driver .....	203
14.2.2	Initialization and de-initialization functions .....	204
14.2.3	IO operation functions .....	204
14.2.4	State and Errors functions .....	205
14.2.5	HAL_DMA_Init .....	205
14.2.6	HAL_DMA_Delinit .....	205
14.2.7	HAL_DMA_Start .....	205
14.2.8	HAL_DMA_Start_IT .....	206

14.2.9	HAL_DMA_Abort .....	206
14.2.10	HAL_DMA_PollForTransfer.....	207
14.2.11	HAL_DMA_IRQHandler.....	207
14.2.12	HAL_DMA_GetState .....	208
14.2.13	HAL_DMA_GetError.....	208
14.3	DMA Firmware driver defines.....	208
14.3.1	DMA.....	208
<b>15</b>	<b>HAL DMA Extension Driver.....</b>	<b>213</b>
15.1	DMAEx Firmware driver defines.....	213
15.1.1	DMAEx.....	213
<b>16</b>	<b>HAL FLASH Generic Driver.....</b>	<b>220</b>
16.1	FLASH Firmware driver registers structures .....	220
16.1.1	FLASH_EraselInitTypeDef .....	220
16.1.2	FLASH_OBProgramInitTypeDef .....	220
16.1.3	FLASH_ProcessTypeDef .....	221
16.2	FLASH Firmware driver API description.....	221
16.2.1	FLASH peripheral features .....	221
16.2.2	How to use this driver .....	222
16.2.3	IO operation functions .....	222
16.2.4	Peripheral Control functions .....	222
16.2.5	Peripheral Errors functions .....	223
16.2.6	HAL_FLASH_Program .....	223
16.2.7	HAL_FLASH_Program_IT .....	223
16.2.8	HAL_FLASH_IRQHandler .....	224
16.2.9	HAL_FLASH_EndOfOperationCallback .....	224
16.2.10	HAL_FLASH_OperationErrorHandler .....	225
16.2.11	HAL_FLASH_Unlock .....	225
16.2.12	HAL_FLASH_Lock .....	225
16.2.13	HAL_FLASH_OB_Unlock.....	226
16.2.14	HAL_FLASH_OB_Lock .....	226
16.2.15	HAL_FLASH_OB_Launch.....	226
16.2.16	HAL_FLASH_GetError .....	226
16.3	FLASH Firmware driver defines .....	227
16.3.1	FLASH .....	227
<b>17</b>	<b>HAL FLASH Extension Driver.....</b>	<b>231</b>
17.1	FLASHEx Firmware driver API description.....	231
17.1.1	Flash peripheral extended features .....	231

17.1.2	How to use this driver .....	231
17.1.3	IO operation functions .....	231
17.1.4	Peripheral Control functions .....	231
17.1.5	HAL_FLASHEx_Erase .....	231
17.1.6	HAL_FLASHEx_Erase_IT .....	232
17.1.7	HAL_FLASHEx_OBErase .....	232
17.1.8	HAL_FLASHEx_OBProgram.....	233
17.1.9	HAL_FLASHEx_OBGetConfig .....	233
17.2	FLASHEx Firmware driver defines .....	233
17.2.1	FLASHEx .....	233
<b>18</b>	<b>HAL GPIO Generic Driver.....</b>	<b>236</b>
18.1	GPIO Firmware driver registers structures .....	236
18.1.1	GPIO_InitTypeDef .....	236
18.2	GPIO Firmware driver API description .....	236
18.2.1	GPIO Peripheral features .....	236
18.2.2	How to use this driver .....	237
18.2.3	Initialization and de-initialization functions .....	237
18.2.4	IO operation functions .....	238
18.2.5	HAL_GPIO_Init.....	238
18.2.6	HAL_GPIO_Delnit .....	238
18.2.7	HAL_GPIO_ReadPin.....	239
18.2.8	HAL_GPIO_WritePin .....	239
18.2.9	HAL_GPIO_TogglePin .....	240
18.2.10	HAL_GPIO_LockPin.....	240
18.2.11	HAL_GPIO_EXTI_IRQHandler .....	241
18.2.12	HAL_GPIO_EXTI_Callback.....	241
18.3	GPIO Firmware driver defines.....	241
18.3.1	GPIO .....	241
<b>19</b>	<b>HAL GPIO Extension Driver.....</b>	<b>245</b>
19.1	GPIOEx Firmware driver defines.....	245
19.1.1	GPIOEx .....	245
<b>20</b>	<b>HAL I2C Generic Driver .....</b>	<b>248</b>
20.1	I2C Firmware driver registers structures .....	248
20.1.1	I2C_InitTypeDef.....	248
20.1.2	I2C_HandleTypeDef.....	248
20.2	I2C Firmware driver API description.....	249
20.2.1	How to use this driver .....	249

20.2.2	Initialization and Configuration functions.....	252
20.2.3	IO operation functions .....	252
20.2.4	Peripheral State and Errors functions .....	254
20.2.5	HAL_I2C_Init .....	254
20.2.6	HAL_I2C_DelInit.....	254
20.2.7	HAL_I2C_MspInit .....	254
20.2.8	HAL_I2C_MspDelInit.....	255
20.2.9	HAL_I2C_Master_Transmit.....	255
20.2.10	HAL_I2C_Master_Receive .....	256
20.2.11	HAL_I2C_Slave_Transmit.....	256
20.2.12	HAL_I2C_Slave_Receive .....	256
20.2.13	HAL_I2C_Master_Transmit_IT.....	257
20.2.14	HAL_I2C_Master_Receive_IT.....	257
20.2.15	HAL_I2C_Slave_Transmit_IT.....	258
20.2.16	HAL_I2C_Slave_Receive_IT.....	258
20.2.17	HAL_I2C_Master_Transmit_DMA.....	259
20.2.18	HAL_I2C_Master_Receive_DMA.....	259
20.2.19	HAL_I2C_Slave_Transmit_DMA.....	259
20.2.20	HAL_I2C_Slave_Receive_DMA.....	260
20.2.21	HAL_I2C_Mem_Write.....	260
20.2.22	HAL_I2C_Mem_Read .....	261
20.2.23	HAL_I2C_Mem_Write_IT .....	261
20.2.24	HAL_I2C_Mem_Read_IT .....	262
20.2.25	HAL_I2C_Mem_Write_DMA .....	262
20.2.26	HAL_I2C_Mem_Read_DMA .....	263
20.2.27	HAL_I2C_IsDeviceReady.....	263
20.2.28	HAL_I2C_EV_IRQHandler .....	264
20.2.29	HAL_I2C_ER_IRQHandler.....	264
20.2.30	HAL_I2C_MasterTxCpltCallback.....	264
20.2.31	HAL_I2C_MasterRxCpltCallback .....	265
20.2.32	HAL_I2C_SlaveTxCpltCallback.....	265
20.2.33	HAL_I2C_SlaveRxCpltCallback .....	265
20.2.34	HAL_I2C_MemTxCpltCallback.....	266
20.2.35	HAL_I2C_MemRxCpltCallback .....	266
20.2.36	HAL_I2C_ErrorCallback .....	267
20.2.37	HAL_I2C_GetState.....	267
20.2.38	HAL_I2C_GetError .....	267
20.3	I2C Firmware driver defines .....	268
20.3.1	I2C .....	268

<b>21 HAL I2C Extension Driver .....</b>	<b>273</b>
21.1    I2CEx Firmware driver API description .....	273
21.1.1    I2C peripheral extension features .....	273
21.1.2    How to use this driver .....	273
21.1.3    Extension features functions .....	273
21.1.4    HAL_I2CEx_AnalogFilter_Config .....	273
21.1.5    HAL_I2CEx_DigitalFilter_Config .....	274
21.1.6    HAL_I2CEx_EnableWakeUp .....	274
21.1.7    HAL_I2CEx_DisableWakeUp .....	274
21.2    I2CEx Firmware driver defines .....	275
21.2.1    I2CEx .....	275
<b>22 HAL I2S Generic Driver .....</b>	<b>276</b>
22.1    I2S Firmware driver registers structures .....	276
22.1.1    I2S_InitTypeDef .....	276
22.1.2    I2S_HandleTypeDef .....	276
22.2    I2S Firmware driver API description .....	277
22.2.1    How to use this driver .....	277
22.2.2    Initialization and de-initialization functions .....	279
22.2.3    IO operation functions .....	279
22.2.4    Peripheral State and Errors functions .....	280
22.2.5    HAL_I2S_Init .....	280
22.2.6    HAL_I2S_DeInit .....	280
22.2.7    HAL_I2S_MspInit .....	281
22.2.8    HAL_I2S_MspDeInit .....	281
22.2.9    HAL_I2S_Transmit .....	281
22.2.10    HAL_I2S_Receive .....	282
22.2.11    HAL_I2S_Transmit_IT .....	283
22.2.12    HAL_I2S_Receive_IT .....	283
22.2.13    HAL_I2S_Transmit_DMA .....	284
22.2.14    HAL_I2S_Receive_DMA .....	284
22.2.15    HAL_I2S_DMAPause .....	285
22.2.16    HAL_I2S_DMAResume .....	285
22.2.17    HAL_I2S_DMAStop .....	286
22.2.18    HAL_I2S_IRQHandler .....	286
22.2.19    HAL_I2S_TxHalfCpltCallback .....	286
22.2.20    HAL_I2S_TxCpltCallback .....	287
22.2.21    HAL_I2S_RxHalfCpltCallback .....	287

22.2.22	HAL_I2S_RxCpltCallback .....	287
22.2.23	HAL_I2S_ErrorCallback .....	288
22.2.24	HAL_I2S_GetState .....	288
22.2.25	HAL_I2S_GetError .....	288
22.3	I2S Firmware driver defines .....	289
22.3.1	I2S .....	289
<b>23</b>	<b>HAL IRDA Generic Driver.....</b>	<b>293</b>
23.1	IRDA Firmware driver registers structures .....	293
23.1.1	IRDA_InitTypeDef.....	293
23.1.2	IRDA_HandleTypeDef .....	293
23.2	IRDA Firmware driver API description.....	294
23.2.1	How to use this driver .....	294
23.2.2	Initialization and Configuration functions.....	296
23.2.3	IO operation functions .....	296
23.2.4	Peripheral Control functions .....	297
23.2.5	HAL_IRDA_Init .....	297
23.2.6	HAL_IRDA_DeInit.....	298
23.2.7	HAL_IRDA_MspInit .....	298
23.2.8	HAL_IRDA_MspDeInit.....	298
23.2.9	HAL_IRDA_Transmit.....	299
23.2.10	HAL_IRDA_Receive .....	299
23.2.11	HAL_IRDA_Transmit_IT.....	300
23.2.12	HAL_IRDA_Receive_IT.....	300
23.2.13	HAL_IRDA_Transmit_DMA.....	300
23.2.14	HAL_IRDA_Receive_DMA.....	301
23.2.15	HAL_IRDA_IRQHandler.....	301
23.2.16	HAL_IRDA_TxCpltCallback.....	301
23.2.17	HAL_IRDA_RxCpltCallback .....	302
23.2.18	HAL_IRDA_ErrorCallback .....	302
23.2.19	HAL_IRDA_GetState .....	302
23.2.20	HAL_IRDA_GetError .....	303
23.3	IRDA Firmware driver defines .....	303
23.3.1	IRDA .....	303
<b>24</b>	<b>HAL IRDA Extension Driver.....</b>	<b>311</b>
24.1	IRDAEx Firmware driver defines .....	311
24.1.1	IRDAEx .....	311
<b>25</b>	<b>HAL IWDG Generic Driver.....</b>	<b>312</b>

25.1	IWDG Firmware driver registers structures .....	312
25.1.1	IWDG_InitTypeDef .....	312
25.1.2	IWDG_HandleTypeDef.....	312
25.2	IWDG Firmware driver API description .....	312
25.2.1	IWDG Specific features .....	312
25.2.2	How to use this driver.....	313
25.2.3	Initialization functions .....	314
25.2.4	IO operation functions .....	314
25.2.5	Peripheral State functions .....	314
25.2.6	HAL_IWDG_Init.....	314
25.2.7	HAL_IWDG_MspInit .....	315
25.2.8	HAL_IWDG_Start .....	315
25.2.9	HAL_IWDG_Refresh.....	315
25.2.10	HAL_IWDG_GetState.....	316
25.3	IWDG Firmware driver defines .....	316
25.3.1	IWDG .....	316
<b>26</b>	<b>HAL PCD Generic Driver .....</b>	<b>319</b>
26.1	PCD Firmware driver registers structures .....	319
26.1.1	PCD_InitTypeDef.....	319
26.1.2	PCD_EPTypeDef.....	319
26.1.3	PCD_HandleTypeDef .....	320
26.2	PCD Firmware driver API description.....	321
26.2.1	How to use this driver .....	321
26.2.2	Initialization and de-initialization functions .....	321
26.2.3	IO operation functions .....	322
26.2.4	Peripheral Control functions .....	322
26.2.5	Peripheral State functions .....	322
26.2.6	HAL_PCD_Init .....	322
26.2.7	HAL_PCD_DelInit.....	323
26.2.8	HAL_PCD_MspInit .....	323
26.2.9	HAL_PCD_MspDelInit.....	323
26.2.10	HAL_PCD_Start .....	324
26.2.11	HAL_PCD_Stop.....	324
26.2.12	HAL_PCD_IRQHandler.....	324
26.2.13	HAL_PCD_DataOutStageCallback .....	325
26.2.14	HAL_PCD_DataInStageCallback .....	325
26.2.15	HAL_PCD_SetupStageCallback .....	325

26.2.16	HAL_PCD_SOFCallback.....	326
26.2.17	HAL_PCD_ResetCallback.....	326
26.2.18	HAL_PCD_SuspendCallback.....	326
26.2.19	HAL_PCD_ResumeCallback.....	327
26.2.20	HAL_PCD_ISOOUTIncompleteCallback.....	327
26.2.21	HAL_PCD_ISOINIncompleteCallback.....	327
26.2.22	HAL_PCD_ConnectCallback.....	328
26.2.23	HAL_PCD_DisconnectCallback .....	328
26.2.24	HAL_PCD_DevConnect .....	328
26.2.25	HAL_PCD_DevDisconnect.....	329
26.2.26	HAL_PCD_SetAddress .....	329
26.2.27	HAL_PCD_EP_Open .....	329
26.2.28	HAL_PCD_EP_Close .....	330
26.2.29	HAL_PCD_EP_Receive .....	330
26.2.30	HAL_PCD_EP_GetRxCount .....	331
26.2.31	HAL_PCD_EP_Transmit .....	331
26.2.32	HAL_PCD_EP_SetStall.....	331
26.2.33	HAL_PCD_EP_ClrStall.....	332
26.2.34	HAL_PCD_EP_Flush .....	332
26.2.35	HAL_PCD_ActiveRemoteWakeup .....	332
26.2.36	HAL_PCD_DeActiveRemoteWakeup.....	333
26.2.37	HAL_PCD_GetState.....	333
26.3	PCD Firmware driver defines .....	334
26.3.1	PCD .....	334
<b>27</b>	<b>HAL PCD Extension Driver .....</b>	<b>343</b>
27.1	PCDEx Firmware driver API description .....	343
27.1.1	Peripheral extended features methods .....	343
27.1.2	HAL_PCDEx_PMAConfig .....	343
<b>28</b>	<b>HAL PWR Generic Driver .....</b>	<b>344</b>
28.1	PWR Firmware driver API description .....	344
28.1.1	Initialization and de-initialization functions .....	344
28.1.2	Peripheral Control functions .....	344
28.1.3	HAL_PWR_EnableBkUpAccess .....	346
28.1.4	HAL_PWR_DisableBkUpAccess.....	346
28.1.5	HAL_PWR_EnableWakeUpPin.....	347
28.1.6	HAL_PWR_DisableWakeUpPin.....	347
28.1.7	HAL_PWR_EnterSLEEPMode.....	347
28.1.8	HAL_PWR_EnterSTOPMode.....	348

28.1.9	HAL_PWR_EnterSTANDBYMode .....	348
28.1.10	HAL_PWR_EnableSleepOnExit.....	349
28.1.11	HAL_PWR_DisableSleepOnExit .....	349
28.1.12	HAL_PWR_EnableSEVOnPend .....	350
28.1.13	HAL_PWR_DisableSEVOnPend.....	350
28.1.14	HAL_PWR_EnableBkUpAccess .....	350
28.1.15	HAL_PWR_DisableBkUpAccess.....	351
28.2	PWR Firmware driver defines .....	351
28.2.1	PWR .....	351
<b>29</b>	<b>HAL PWR Extension Driver .....</b>	<b>353</b>
29.1	PWREx Firmware driver registers structures .....	353
29.1.1	PWR_PVDTTypeDef .....	353
29.2	PWREx Firmware driver API description.....	353
29.2.1	Peripheral extended control functions .....	353
29.2.2	HAL_PWR_PVDConfig .....	354
29.2.3	HAL_PWR_EnablePVD.....	354
29.2.4	HAL_PWR_DisablePVD.....	354
29.2.5	HAL_PWR_PVD_IRQHandler.....	355
29.2.6	HAL_PWR_PVDCallback.....	355
29.2.7	HAL_PWR_EnableVddio2Monitor.....	355
29.2.8	HAL_PWR_DisableVddio2Monitor.....	356
29.2.9	HAL_PWR_Vddio2Monitor_IRQHandler.....	356
29.2.10	HAL_PWR_Vddio2MonitorCallback.....	356
29.3	PWREx Firmware driver defines .....	356
29.3.1	PWREx .....	356
<b>30</b>	<b>HAL RCC Generic Driver.....</b>	<b>361</b>
30.1	RCC Firmware driver registers structures .....	361
30.1.1	RCC_PLLInitTypeDef .....	361
30.1.2	RCC_OscInitTypeDef .....	361
30.1.3	RCC_ClkInitTypeDef .....	362
30.2	RCC Firmware driver API description .....	362
30.2.1	RCC specific features.....	362
30.2.2	RCC Limitations.....	363
30.2.3	Initialization and de-initialization function .....	363
30.2.4	Peripheral Control function.....	364
30.2.5	HAL_RCC_DeInit .....	365
30.2.6	HAL_RCC_OscConfig .....	365

30.2.7	HAL_RCC_ClockConfig .....	365
30.2.8	HAL_RCC_MCOConfig .....	366
30.2.9	HAL_RCC_EnableCSS .....	367
30.2.10	HAL_RCC_DisableCSS .....	367
30.2.11	HAL_RCC_GetSysClockFreq .....	368
30.2.12	HAL_RCC_GetHCLKFreq .....	368
30.2.13	HAL_RCC_GetPCLK1Freq .....	369
30.2.14	HAL_RCC_GetOscConfig .....	369
30.2.15	HAL_RCC_GetClockConfig .....	369
30.2.16	HAL_RCC_NMI_IRQHandler .....	370
30.2.17	HAL_RCC_CCSCallback .....	370
30.3	RCC Firmware driver defines .....	370
30.3.1	RCC .....	370
<b>31</b>	<b>HAL RCC Extension Driver .....</b>	<b>387</b>
31.1	RCCEEx Firmware driver registers structures .....	387
31.1.1	RCC_PерiphCLKInitTypeDef .....	387
31.1.2	RCC_CRSInitTypeDef .....	387
31.1.3	RCC_CRSSynchroInfoTypeDef .....	388
31.2	RCCEEx Firmware driver API description .....	388
31.2.1	How to use this driver .....	388
31.2.2	Extended Peripheral Control functions .....	389
31.2.3	HAL_RCC_OscConfig .....	389
31.2.4	HAL_RCC_ClockConfig .....	389
31.2.5	HAL_RCC_GetSysClockFreq .....	390
31.2.6	HAL_RCCEEx_PeriphCLKConfig .....	391
31.2.7	HAL_RCCEEx_GetPeriphCLKConfig .....	391
31.2.8	HAL_RCCEEx_CRSConfig .....	392
31.2.9	HAL_RCCEEx_CRSSoftwareSynchronizationGenerate .....	392
31.2.10	HAL_RCCEEx_CRSGetSynchronizationInfo .....	392
31.2.11	HAL_RCCEEx_CRSWaitSynchronization .....	393
31.3	RCCEEx Firmware driver defines .....	393
31.3.1	RCCEEx .....	393
<b>32</b>	<b>HAL RTC Generic Driver .....</b>	<b>405</b>
32.1	RTC Firmware driver registers structures .....	405
32.1.1	RTC_InitTypeDef .....	405
32.1.2	RTC_TimeTypeDef .....	405
32.1.3	RTC_DateTypeDef .....	406
32.1.4	RTC_AlarmTypeDef .....	406

32.1.5	RTC_HandleTypeDef .....	407
32.2	RTC Firmware driver API description.....	407
32.2.1	RTC Operating Condition .....	407
32.2.2	Backup Domain Reset.....	408
32.2.3	Backup Domain Access.....	408
32.2.4	How to use RTC Driver.....	408
32.2.5	RTC and low power modes .....	409
32.2.6	Initialization and de-initialization functions .....	409
32.2.7	RTC Time and Date functions .....	409
32.2.8	RTC Alarm functions .....	410
32.2.9	Peripheral Control functions .....	410
32.2.10	Peripheral State functions .....	410
32.2.11	HAL_RTC_Init .....	410
32.2.12	HAL_RTC_DelInit.....	411
32.2.13	HAL_RTC_MspInit.....	411
32.2.14	HAL_RTC_MspDeInit.....	411
32.2.15	HAL_RTC_SetTime.....	412
32.2.16	HAL_RTC_GetTime .....	412
32.2.17	HAL_RTC_SetDate .....	412
32.2.18	HAL_RTC_GetDate .....	413
32.2.19	HAL_RTC_SetAlarm .....	413
32.2.20	HAL_RTC_SetAlarm_IT .....	414
32.2.21	HAL_RTC_DeactivateAlarm.....	414
32.2.22	HAL_RTC_GetAlarm.....	415
32.2.23	HAL_RTC_AlarmIRQHandler.....	415
32.2.24	HAL_RTC_AlarmAEventCallback .....	416
32.2.25	HAL_RTC_PollForAlarmAEvent.....	416
32.2.26	HAL_RTC_WaitForSynchro .....	416
32.2.27	HAL_RTC_GetState .....	417
32.3	RTC Firmware driver defines .....	417
32.3.1	RTC .....	417
33	<b>HAL RTC Extension Driver .....</b>	<b>426</b>
33.1	RTCEx Firmware driver registers structures .....	426
33.1.1	RTC_TamperTypeDef .....	426
33.2	RTCEx Firmware driver API description.....	426
33.2.1	How to use this driver .....	426
33.2.2	RTCTimeStamp and Tamper functions.....	427
33.2.3	RTC Wake-up functions .....	428

33.2.4	Extension Peripheral Control functions .....	428
33.2.5	HAL_RTCEx_SetTimeStamp .....	428
33.2.6	HAL_RTCEx_SetTimeStamp_IT.....	429
33.2.7	HAL_RTCEx_DeactivateTimeStamp .....	429
33.2.8	HAL_RTCEx_GetTimeStamp.....	430
33.2.9	HAL_RTCEx_SetTamper .....	430
33.2.10	HAL_RTCEx_SetTamper_IT .....	431
33.2.11	HAL_RTCEx_DeactivateTamper .....	431
33.2.12	HAL_RTCEx_TamperTimeStampIRQHandler .....	431
33.2.13	HAL_RTCEx_TimeStampEventCallback .....	432
33.2.14	HAL_RTCEx_Tamper1EventCallback .....	432
33.2.15	HAL_RTCEx_Tamper2EventCallback .....	432
33.2.16	HAL_RTCEx_Tamper3EventCallback .....	433
33.2.17	HAL_RTCEx_PollForTimeStampEvent.....	433
33.2.18	HAL_RTCEx_PollForTamper1Event.....	433
33.2.19	HAL_RTCEx_PollForTamper2Event.....	434
33.2.20	HAL_RTCEx_PollForTamper3Event.....	434
33.2.21	HAL_RTCEx_SetWakeUpTimer .....	434
33.2.22	HAL_RTCEx_SetWakeUpTimer_IT .....	435
33.2.23	HAL_RTCEx_DeactivateWakeUpTimer.....	435
33.2.24	HAL_RTCEx_GetWakeUpTimer .....	436
33.2.25	HAL_RTCEx_WakeUpTimerIRQHandler.....	436
33.2.26	HAL_RTCEx_WakeUpTimerEventCallback.....	436
33.2.27	HAL_RTCEx_PollForWakeUpTimerEvent .....	437
33.2.28	HAL_RTCEx_BKUPWrite.....	437
33.2.29	HAL_RTCEx_BKUPRead .....	437
33.2.30	HAL_RTCEx_SetSmoothCalib.....	438
33.2.31	HAL_RTCEx_SetSynchroShift.....	438
33.2.32	HAL_RTCEx_SetCalibrationOutPut .....	439
33.2.33	HAL_RTCEx_DeactivateCalibrationOutPut .....	439
33.2.34	HAL_RTCEx_SetRefClock .....	440
33.2.35	HAL_RTCEx_DeactivateRefClock .....	440
33.2.36	HAL_RTCEx_EnableBypassShadow .....	440
33.2.37	HAL_RTCEx_DisableBypassShadow .....	441
33.3	RTCEEx Firmware driver defines .....	441
33.3.1	RTCEEx .....	441
<b>34</b>	<b>HAL SMARTCARD Generic Driver.....</b>	<b>451</b>
34.1	SMARTCARD Firmware driver registers structures .....	451

34.1.1	SMARTCARD_InitTypeDef .....	451
34.1.2	SMARTCARD_AdvFeatureInitTypeDef.....	452
34.1.3	SMARTCARD_HandleTypeDef.....	453
34.2	SMARTCARD Firmware driver API description.....	454
34.2.1	How to use this driver .....	454
34.2.2	Initialization and Configuration functions.....	455
34.2.3	IO operation functions .....	456
34.2.4	Peripheral State and Errors functions .....	457
34.2.5	HAL_SMARTCARD_Init.....	458
34.2.6	HAL_SMARTCARD_DelInit .....	458
34.2.7	HAL_SMARTCARD_MspInit .....	458
34.2.8	HAL_SMARTCARD_MspDelInit .....	459
34.2.9	HAL_SMARTCARD_Transmit.....	459
34.2.10	HAL_SMARTCARD_Receive.....	459
34.2.11	HAL_SMARTCARD_Transmit_IT .....	460
34.2.12	HAL_SMARTCARD_Receive_IT .....	460
34.2.13	HAL_SMARTCARD_Transmit_DMA.....	461
34.2.14	HAL_SMARTCARD_Receive_DMA.....	461
34.2.15	HAL_SMARTCARD_IRQHandler.....	461
34.2.16	HAL_SMARTCARD_TxCpltCallback .....	462
34.2.17	HAL_SMARTCARD_RxCpltCallback .....	462
34.2.18	HAL_SMARTCARD_ErrorCallback.....	462
34.2.19	HAL_SMARTCARD_GetState .....	463
34.2.20	HAL_SMARTCARD_GetError.....	463
34.3	SMARTCARD Firmware driver defines .....	464
34.3.1	SMARTCARD .....	464
<b>35</b>	<b>HAL SMARTCARD Extension Driver.....</b>	<b>474</b>
35.1	SMARTCARDEX Firmware driver API description .....	474
35.1.1	Peripheral Control functions .....	474
35.1.2	HAL_SMARTCARDEX_BlockLength_Config .....	474
35.1.3	HAL_SMARTCARDEX_TimeOut_Config .....	474
35.1.4	HAL_SMARTCARDEX_EnableReceiverTimeOut .....	475
35.1.5	HAL_SMARTCARDEX_DisableReceiverTimeOut .....	475
35.2	SMARTCARDEX Firmware driver defines .....	476
35.2.1	SMARTCARDEX.....	476
<b>36</b>	<b>HAL SMBUS Generic Driver.....</b>	<b>477</b>
36.1	SMBUS Firmware driver registers structures .....	477

36.1.1	SMBUS_InitTypeDef .....	477
36.1.2	SMBUS_HandleTypeDef.....	478
36.2	SMBUS Firmware driver API description .....	478
36.2.1	Initialization and de-initialization functions .....	478
36.2.2	IO operation functions .....	479
36.2.3	Peripheral State and Errors functions .....	480
36.2.4	HAL_SMBUS_Init.....	480
36.2.5	HAL_SMBUS_DelInit .....	480
36.2.6	HAL_SMBUS_MspInit .....	481
36.2.7	HAL_SMBUS_MspDeInit.....	481
36.2.8	HAL_SMBUS_Master_Transmit_IT .....	481
36.2.9	HAL_SMBUS_Master_Receive_IT .....	482
36.2.10	HAL_SMBUS_Master_Abort_IT.....	482
36.2.11	HAL_SMBUS_Slave_Transmit_IT .....	483
36.2.12	HAL_SMBUS_Slave_Receive_IT .....	483
36.2.13	HAL_SMBUS_Slave_Listen_IT .....	484
36.2.14	HAL_SMBUS_DisableListen_IT .....	484
36.2.15	HAL_SMBUS_EnableAlert_IT .....	485
36.2.16	HAL_SMBUS_DisableAlert_IT .....	485
36.2.17	HAL_SMBUS_IsDeviceReady .....	485
36.2.18	HAL_SMBUS_EV_IRQHandler.....	486
36.2.19	HAL_SMBUS_ER_IRQHandler.....	486
36.2.20	HAL_SMBUS_MasterTxCpltCallback .....	486
36.2.21	HAL_SMBUS_MasterRxCpltCallback .....	487
36.2.22	HAL_SMBUS_SlaveTxCpltCallback .....	487
36.2.23	HAL_SMBUS_SlaveRxCpltCallback .....	488
36.2.24	HAL_SMBUS_SlaveAddrCallback .....	488
36.2.25	HAL_SMBUS_SlaveListenCpltCallback.....	488
36.2.26	HAL_SMBUS_ErrorCallback.....	489
36.2.27	HAL_SMBUS_GetState.....	489
36.2.28	HAL_SMBUS_GetError .....	489
36.3	SMBUS Firmware driver defines .....	490
36.3.1	SMBUS .....	490
37	HAL SPI Generic Driver.....	497
37.1	SPI Firmware driver registers structures .....	497
37.1.1	SPI_InitTypeDef .....	497
37.1.2	__SPI_HandleTypeDef.....	498
37.2	SPI Firmware driver API description .....	499

37.2.1	How to use this driver .....	499
37.2.2	Initialization and Configuration functions.....	500
37.2.3	IO operation functions .....	500
37.2.4	Peripheral Control functions .....	501
37.2.5	HAL_SPI_Init.....	502
37.2.6	HAL_SPI_DeInit .....	502
37.2.7	HAL_SPI_MspInit .....	502
37.2.8	HAL_SPI_MspDeInit.....	503
37.2.9	HAL_SPI_InitExtended.....	503
37.2.10	HAL_SPI_Transmit.....	503
37.2.11	HAL_SPI_Receive.....	504
37.2.12	HAL_SPI_TransmitReceive.....	504
37.2.13	HAL_SPI_Transmit_IT.....	505
37.2.14	HAL_SPI_Receive_IT.....	505
37.2.15	HAL_SPI_TransmitReceive_IT .....	505
37.2.16	HAL_SPI_Transmit_DMA.....	506
37.2.17	HAL_SPI_Receive_DMA.....	506
37.2.18	HAL_SPI_TransmitReceive_DMA.....	507
37.2.19	HAL_SPI_DMAPause.....	507
37.2.20	HAL_SPI_DMAResume .....	507
37.2.21	HAL_SPI_DMAStop .....	508
37.2.22	HAL_SPI_IRQHandler.....	508
37.2.23	HAL_SPI_FlushRxFifo.....	508
37.2.24	HAL_SPI_TxCpltCallback .....	509
37.2.25	HAL_SPI_RxCpltCallback .....	509
37.2.26	HAL_SPI_TxRxCpltCallback .....	509
37.2.27	HAL_SPI_TxHalfCpltCallback .....	510
37.2.28	HAL_SPI_RxHalfCpltCallback.....	510
37.2.29	HAL_SPI_TxRxHalfCpltCallback.....	510
37.2.30	HAL_SPI_ErrorCallback .....	511
37.2.31	HAL_SPI_GetState.....	511
37.2.32	HAL_SPI_GetError .....	512
37.3	SPI Firmware driver defines .....	512
37.3.1	SPI.....	512
<b>38</b>	<b>HAL TIM Generic Driver .....</b>	<b>519</b>
38.1	TIM Firmware driver registers structures.....	519
38.1.1	TIM_Base_InitTypeDef.....	519
38.1.2	TIM_OC_InitTypeDef.....	519

38.1.3	TIM_OnePulse_InitTypeDef .....	520
38.1.4	TIM_IC_InitTypeDef .....	521
38.1.5	TIM_Encoder_InitTypeDef .....	521
38.1.6	TIM_ClockConfigTypeDef .....	522
38.1.7	TIM_ClearInputConfigTypeDef.....	522
38.1.8	TIM_SlaveConfigTypeDef .....	523
38.1.9	TIM_HandleTypeDef .....	523
38.2	TIM Firmware driver API description .....	524
38.2.1	TIMER Generic features.....	524
38.2.2	How to use this driver .....	524
38.2.3	Time Base functions .....	525
38.2.4	Time Output Compare functions .....	525
38.2.5	Time PWM functions .....	526
38.2.6	Time Input Capture functions .....	526
38.2.7	Time One Pulse functions .....	527
38.2.8	Time Encoder functions.....	527
38.2.9	IRQ handler management .....	528
38.2.10	Peripheral Control functions .....	528
38.2.11	TIM Callbacks functions .....	528
38.2.12	Peripheral State functions .....	529
38.2.13	HAL_TIM_Base_Init .....	529
38.2.14	HAL_TIM_Base_DeInit.....	529
38.2.15	HAL_TIM_Base_MspInit.....	529
38.2.16	HAL_TIM_Base_MspDeInit.....	530
38.2.17	HAL_TIM_Base_Start.....	530
38.2.18	HAL_TIM_Base_Stop.....	530
38.2.19	HAL_TIM_Base_Start_IT .....	531
38.2.20	HAL_TIM_Base_Stop_IT.....	531
38.2.21	HAL_TIM_Base_Start_DMA .....	531
38.2.22	HAL_TIM_Base_Stop_DMA.....	532
38.2.23	HAL_TIM_OC_Init .....	532
38.2.24	HAL_TIM_OC_DeInit.....	532
38.2.25	HAL_TIM_OC_MspInit .....	533
38.2.26	HAL_TIM_OC_MspDeInit.....	533
38.2.27	HAL_TIM_OC_Start .....	533
38.2.28	HAL_TIM_OC_Stop.....	534
38.2.29	HAL_TIM_OC_Start_IT .....	534
38.2.30	HAL_TIM_OC_Stop_IT .....	535
38.2.31	HAL_TIM_OC_Start_DMA .....	535

38.2.32	HAL_TIM_OC_Stop_DMA .....	536
38.2.33	HAL_TIM_PWM_Init.....	536
38.2.34	HAL_TIM_PWM_DelInit .....	537
38.2.35	HAL_TIM_PWM_MspInit.....	537
38.2.36	HAL_TIM_PWM_MspDelInit .....	537
38.2.37	HAL_TIM_PWM_Start.....	538
38.2.38	HAL_TIM_PWM_Stop .....	538
38.2.39	HAL_TIM_PWM_Start_IT .....	538
38.2.40	HAL_TIM_PWM_Stop_IT .....	539
38.2.41	HAL_TIM_PWM_Start_DMA.....	539
38.2.42	HAL_TIM_PWM_Stop_DMA .....	540
38.2.43	HAL_TIM_IC_Init .....	540
38.2.44	HAL_TIM_IC_DelInit .....	541
38.2.45	HAL_TIM_IC_MspInit .....	541
38.2.46	HAL_TIM_IC_MspDelInit.....	541
38.2.47	HAL_TIM_IC_Start .....	542
38.2.48	HAL_TIM_IC_Stop .....	542
38.2.49	HAL_TIM_IC_Start_IT .....	543
38.2.50	HAL_TIM_IC_Stop_IT .....	543
38.2.51	HAL_TIM_IC_Start_DMA .....	543
38.2.52	HAL_TIM_IC_Stop_DMA .....	544
38.2.53	HAL_TIM_OnePulse_Init.....	544
38.2.54	HAL_TIM_OnePulse_DelInit .....	545
38.2.55	HAL_TIM_OnePulse_MspInit.....	545
38.2.56	HAL_TIM_OnePulse_MspDelInit .....	546
38.2.57	HAL_TIM_OnePulse_Start.....	546
38.2.58	HAL_TIM_OnePulse_Stop .....	546
38.2.59	HAL_TIM_OnePulse_Start_IT.....	547
38.2.60	HAL_TIM_OnePulse_Stop_IT .....	547
38.2.61	HAL_TIM_Encoder_Init .....	548
38.2.62	HAL_TIM_Encoder_DelInit .....	548
38.2.63	HAL_TIM_Encoder_MspInit .....	548
38.2.64	HAL_TIM_Encoder_MspDelInit.....	549
38.2.65	HAL_TIM_Encoder_Start .....	549
38.2.66	HAL_TIM_Encoder_Stop .....	549
38.2.67	HAL_TIM_Encoder_Start_IT .....	550
38.2.68	HAL_TIM_Encoder_Stop_IT .....	550
38.2.69	HAL_TIM_Encoder_Start_DMA .....	551

38.2.70	HAL_TIM_Encoder_Stop_DMA .....	551
38.2.71	HAL_TIM_IRQHandler .....	551
38.2.72	HAL_TIM_OC_ConfigChannel .....	552
38.2.73	HAL_TIM_IC_ConfigChannel .....	552
38.2.74	HAL_TIM_PWM_ConfigChannel .....	553
38.2.75	HAL_TIM_OnePulse_ConfigChannel .....	553
38.2.76	HAL_TIM_DMABurst_WriteStart .....	554
38.2.77	HAL_TIM_DMABurst_WriteStop .....	555
38.2.78	HAL_TIM_DMABurst_ReadStart .....	555
38.2.79	HAL_TIM_DMABurst_ReadStop .....	556
38.2.80	HAL_TIM_GenerateEvent .....	557
38.2.81	HAL_TIM_ConfigOCrefClear .....	557
38.2.82	HAL_TIM_ConfigClockSource .....	558
38.2.83	HAL_TIM_ConfigTI1Input .....	558
38.2.84	HAL_TIM_SlaveConfigSynchronization .....	559
38.2.85	HAL_TIM_SlaveConfigSynchronization_IT .....	559
38.2.86	HAL_TIM_ReadCapturedValue .....	560
38.2.87	HAL_TIM_PeriodElapsedCallback .....	560
38.2.88	HAL_TIM_OC_DelayElapsedCallback .....	560
38.2.89	HAL_TIM_IC_CaptureCallback .....	561
38.2.90	HAL_TIM_PWM_PulseFinishedCallback .....	561
38.2.91	HAL_TIM_TriggerCallback .....	561
38.2.92	HAL_TIM_ErrorCallback .....	562
38.2.93	HAL_TIM_Base_GetState .....	562
38.2.94	HAL_TIM_OC_GetState .....	562
38.2.95	HAL_TIM_PWM_GetState .....	563
38.2.96	HAL_TIM_IC_GetState .....	563
38.2.97	HAL_TIM_OnePulse_GetState .....	563
38.2.98	HAL_TIM_Encoder_GetState .....	564
38.3	TIM Firmware driver defines .....	564
38.3.1	TIM .....	564
<b>39</b>	<b>HAL TIM Extension Driver .....</b>	<b>579</b>
39.1	TIMEx Firmware driver registers structures .....	579
39.1.1	TIM_HallSensor_InitTypeDef .....	579
39.1.2	TIM_MasterConfigTypeDef .....	579
39.1.3	TIM_BreakDeadTimeConfigTypeDef .....	579
39.2	TIMEx Firmware driver API description .....	580
39.2.1	TIMER Extended features .....	580

39.2.2	How to use this driver .....	580
39.2.3	Timer Hall Sensor functions .....	581
39.2.4	Timer Complementary Output Compare functions.....	582
39.2.5	Timer Complementary PWM functions.....	582
39.2.6	Timer Complementary One Pulse functions.....	582
39.2.7	Peripheral Control functions .....	583
39.2.8	HAL_TIMEx_HallSensor_Init.....	583
39.2.9	HAL_TIMEx_HallSensor_DelInit .....	583
39.2.10	HAL_TIMEx_HallSensor_MspInit.....	584
39.2.11	HAL_TIMEx_HallSensor_MspDeInit .....	584
39.2.12	HAL_TIMEx_HallSensor_Start.....	584
39.2.13	HAL_TIMEx_HallSensor_Stop .....	585
39.2.14	HAL_TIMEx_HallSensor_Start_IT.....	585
39.2.15	HAL_TIMEx_HallSensor_Stop_IT .....	585
39.2.16	HAL_TIMEx_HallSensor_Start_DMA.....	586
39.2.17	HAL_TIMEx_HallSensor_Stop_DMA .....	586
39.2.18	HAL_TIMEx_OCN_Start.....	586
39.2.19	HAL_TIMEx_OCN_Stop .....	587
39.2.20	HAL_TIMEx_OCN_Start_IT .....	587
39.2.21	HAL_TIMEx_OCN_Stop_IT .....	588
39.2.22	HAL_TIMEx_OCN_Start_DMA .....	588
39.2.23	HAL_TIMEx_OCN_Stop_DMA .....	589
39.2.24	HAL_TIMEx_PWMN_Start .....	589
39.2.25	HAL_TIMEx_PWMN_Stop .....	590
39.2.26	HAL_TIMEx_PWMN_Start_IT .....	590
39.2.27	HAL_TIMEx_PWMN_Stop_IT .....	591
39.2.28	HAL_TIMEx_PWMN_Start_DMA .....	591
39.2.29	HAL_TIMEx_PWMN_Stop_DMA .....	592
39.2.30	HAL_TIMEx_OnePulseN_Start .....	592
39.2.31	HAL_TIMEx_OnePulseN_Stop .....	593
39.2.32	HAL_TIMEx_OnePulseN_Start_IT .....	593
39.2.33	HAL_TIMEx_OnePulseN_Stop_IT .....	593
39.2.34	HAL_TIMEx_ConfigCommutationEvent .....	594
39.2.35	HAL_TIMEx_ConfigCommutationEvent_IT .....	595
39.2.36	HAL_TIMEx_ConfigCommutationEvent_DMA .....	595
39.2.37	HAL_TIMEx_MasterConfigSynchronization .....	596
39.2.38	HAL_TIMEx_ConfigBreakDeadTime.....	597
39.2.39	HAL_TIMEx_RemapConfig .....	597

39.2.40	HAL_TIMEx_CommutationCallback .....	597
39.2.41	HAL_TIMEx_BreakCallback .....	598
39.2.42	HAL_TIMEx_DMACommutationCplt .....	598
39.2.43	HAL_TIMEx_HallSensor_GetState .....	598
39.3	TIMEx Firmware driver defines .....	599
39.3.1	TIME .....	599
<b>40</b>	<b>HAL TSC Generic Driver .....</b>	<b>600</b>
40.1	TSC Firmware driver registers structures.....	600
40.1.1	TSC_InitTypeDef .....	600
40.1.2	TSC_IOConfigTypeDef.....	600
40.1.3	TSC_HandleTypeDef .....	601
40.2	TSC Firmware driver API description .....	601
40.2.1	TSC specific features .....	601
40.2.2	How to use this driver .....	601
40.2.3	Initialization and de-initialization functions .....	602
40.2.4	Peripheral Control functions .....	602
40.2.5	State functions.....	602
40.2.6	HAL_TSC_Init.....	603
40.2.7	HAL_TSC_DelInit .....	603
40.2.8	HAL_TSC_MspInit.....	603
40.2.9	HAL_TSC_MspDeInit .....	604
40.2.10	HAL_TSC_Start.....	604
40.2.11	HAL_TSC_Start_IT.....	604
40.2.12	HAL_TSC_Stop .....	605
40.2.13	HAL_TSC_Stop_IT.....	605
40.2.14	HAL_TSC_GroupGetStatus .....	605
40.2.15	HAL_TSC_GroupGetValue .....	606
40.2.16	HAL_TSC_IOConfig .....	606
40.2.17	HAL_TSC_IODischarge .....	607
40.2.18	HAL_TSC_GetState .....	607
40.2.19	HAL_TSC_PollForAcquisition .....	607
40.2.20	HAL_TSC_IRQHandler .....	608
40.2.21	HAL_TSC_ConvCpltCallback.....	608
40.2.22	HAL_TSC_ErrorCallback.....	608
40.3	TSC Firmware driver defines.....	609
40.3.1	TSC.....	609
<b>41</b>	<b>HAL UART Generic Driver.....</b>	<b>618</b>
41.1	UART Firmware driver registers structures .....	618

41.1.1	UART_InitTypeDef .....	618
41.1.2	UART_AdvFeatureInitTypeDef.....	619
41.1.3	UART_WakeUpTypeDef .....	619
41.1.4	UART_HandleTypeDef.....	620
41.2	UART Firmware driver API description .....	621
41.2.1	How to use this driver .....	621
41.2.2	Initialization and Configuration functions.....	623
41.2.3	IO operation functions .....	624
41.2.4	Peripheral Control functions .....	625
41.2.5	HAL_UART_Init.....	625
41.2.6	HAL_HalfDuplex_Init .....	626
41.2.7	HAL_MultiProcessor_Init.....	626
41.2.8	HAL_UART_DeInit .....	627
41.2.9	HAL_UART_MspInit .....	627
41.2.10	HAL_UART_MspDeInit.....	627
41.2.11	HAL_UART_Transmit.....	628
41.2.12	HAL_UART_Receive.....	628
41.2.13	HAL_UART_Transmit_IT.....	628
41.2.14	HAL_UART_Receive_IT.....	629
41.2.15	HAL_UART_Transmit_DMA.....	629
41.2.16	HAL_UART_Receive_DMA.....	630
41.2.17	HAL_UART_DMAPause.....	630
41.2.18	HAL_UART_DMAResume .....	630
41.2.19	HAL_UART_DMAStop .....	631
41.2.20	HAL_UART_TxCpltCallback .....	631
41.2.21	HAL_UART_TxHalfCpltCallback .....	631
41.2.22	HAL_UART_RxCpltCallback .....	632
41.2.23	HAL_UART_RxHalfCpltCallback .....	632
41.2.24	HAL_UART_ErrorCallback.....	632
41.2.25	HAL_MultiProcessor_EnableMuteMode .....	633
41.2.26	HAL_MultiProcessor_DisableMuteMode.....	633
41.2.27	HAL_MultiProcessor_EnterMuteMode .....	633
41.2.28	HAL_HalfDuplex_EnableTransmitter .....	634
41.2.29	HAL_HalfDuplex_EnableReceiver .....	634
41.2.30	HAL_UART_GetState.....	634
41.2.31	HAL_UART_GetError .....	635
41.3	UART Firmware driver defines .....	635
41.3.1	UART .....	635

<b>42 HAL UART Extension Driver .....</b>	<b>648</b>
42.1    UARTEEx Firmware driver API description .....	648
42.1.1    Initialization and Configuration functions.....	648
42.1.2    IO operation function .....	648
42.1.3    Peripheral Control function.....	649
42.1.4    HAL_RS485Ex_Init.....	649
42.1.5    HAL_LIN_Init .....	650
42.1.6    HAL_UART_IRQHandler.....	650
42.1.7    HAL_UART_WakeupCallback.....	651
42.1.8    HAL_UARTEEx_StopModeWakeUpSourceConfig .....	651
42.1.9    HAL_UARTEEx_EnableStopMode.....	651
42.1.10    HAL_UARTEEx_DisableStopMode .....	652
42.1.11    HAL_MultiProcessorEx_AddressLength_Set.....	652
42.1.12    HAL_LIN_SendBreak .....	652
42.2    UARTEEx Firmware driver defines .....	653
42.2.1    UARTEEx.....	653
<b>43 HAL USART Generic Driver .....</b>	<b>655</b>
43.1    USART Firmware driver registers structures.....	655
43.1.1    USART_InitTypeDef .....	655
43.1.2    USART_HandleTypeDef .....	655
43.2    USART Firmware driver API description .....	656
43.2.1    How to use this driver .....	656
43.2.2    Initialization and Configuration functions.....	658
43.2.3    IO operation functions .....	659
43.2.4    Peripheral Control functions .....	660
43.2.5    HAL_USART_Init.....	660
43.2.6    HAL_USART_DeInit.....	661
43.2.7    HAL_USART_MspInit.....	661
43.2.8    HAL_USART_MspDeInit .....	661
43.2.9    HAL_USART_CheckIdleState .....	662
43.2.10    HAL_USART_Transmit .....	662
43.2.11    HAL_USART_Receive .....	662
43.2.12    HAL_USART_TransmitReceive .....	663
43.2.13    HAL_USART_Transmit_IT .....	663
43.2.14    HAL_USART_Receive_IT .....	664
43.2.15    HAL_USART_TransmitReceive_IT .....	664
43.2.16    HAL_USART_Transmit_DMA .....	665
43.2.17    HAL_USART_Receive_DMA .....	665

43.2.18	HAL_USART_TransmitReceive_DMA .....	665
43.2.19	HAL_USART_DMAPause .....	666
43.2.20	HAL_USART_DMAResume .....	666
43.2.21	HAL_USART_DMAStop .....	667
43.2.22	HAL_USART_IRQHandler .....	667
43.2.23	HAL_USART_TxCpltCallback .....	667
43.2.24	HAL_USART_TxHalfCpltCallback .....	668
43.2.25	HAL_USART_RxCpltCallback .....	668
43.2.26	HAL_USART_RxHalfCpltCallback .....	668
43.2.27	HAL_USART_TxRxCpltCallback .....	669
43.2.28	HAL_USART_ErrorCallback .....	669
43.2.29	HAL_USART_GetState .....	669
43.2.30	HAL_USART_GetError .....	670
43.3	USART Firmware driver defines .....	670
43.3.1	USART .....	670
<b>44</b>	<b>HAL USART Extension Driver .....</b>	<b>677</b>
44.1	USARTEEx Firmware driver defines .....	677
44.1.1	USARTEEx .....	677
<b>45</b>	<b>HAL WWDG Generic Driver .....</b>	<b>678</b>
45.1	WWDG Firmware driver registers structures .....	678
45.1.1	WWDG_InitTypeDef .....	678
45.1.2	WWDG_HandleTypeDef .....	678
45.2	WWDG Firmware driver API description .....	678
45.2.1	WWDG specific features .....	678
45.2.2	How to use this driver .....	679
45.2.3	Initialization and de-initialization functions .....	679
45.2.4	IO operation functions .....	680
45.2.5	Peripheral State functions .....	680
45.2.6	HAL_WWDG_Init .....	680
45.2.7	HAL_WWDG_DeInit .....	681
45.2.8	HAL_WWDG_MspInit .....	681
45.2.9	HAL_WWDG_MspDeInit .....	681
45.2.10	HAL_WWDG_WakeupCallback .....	682
45.2.11	HAL_WWDG_Start .....	682
45.2.12	HAL_WWDG_Start_IT .....	682
45.2.13	HAL_WWDG_Refresh .....	683
45.2.14	HAL_WWDG_IRQHandler .....	683

---

45.2.15	HAL_WWDG_WakeupCallback .....	684
45.2.16	HAL_WWDG_GetState .....	684
45.3	WWDG Firmware driver defines.....	684
45.3.1	WWDG.....	684
<b>46</b>	<b>FAQs.....</b>	<b>687</b>
<b>47</b>	<b>Revision history .....</b>	<b>691</b>

## List of tables

Table 1: Acronyms and definitions.....	35
Table 2: HAL drivers files.....	37
Table 3: User-application files .....	38
Table 4: APis classification .....	43
Table 5: List of devices supported by HAL drivers .....	43
Table 6: HAL API naming rules .....	45
Table 7: Macros handling interrupts and specific clock configurations .....	46
Table 8: Callback functions.....	47
Table 9: HAL generic APIs .....	48
Table 10: HAL extension APIs.....	49
Table 11: Define statements used for HAL configuration .....	53
Table 12: Description of GPIO_InitTypeDef structure .....	55
Table 13: Description of EXTI configuration macros .....	57
Table 14: MSP functions.....	62
Table 15: Timeout values .....	66
Table 16: COMP Inputs for STM32F05xx, STM32F07x and STM32F09x devices .....	147
Table 17: COMP outputs for STM32F05xx, STM32F07x and STM32F09x devices.....	147
Table 18: Redirection of COMP outputs to embedded timers for STM32F05xx, STM32F07x and STM32F09x devices .....	148
Table 19: IRDA frame formats .....	296
Table 20: Number of wait states (WS) according to system clock (SYSCLK) frequency.....	364
Table 21: USART frame formats .....	456
Table 22: Maximum SPI frequency vs data size .....	499
Table 23: UART frame formats.....	623
Table 24: USART frame formats .....	659
Table 25: Document revision history .....	691

## **List of figures**

Figure 1: Example of project template .....	40
Figure 2: Adding device-specific functions .....	50
Figure 3: Adding family-specific functions .....	50
Figure 4: Adding new peripherals .....	51
Figure 5: Updating existing APIs .....	51
Figure 6: File inclusion model .....	52
Figure 7: HAL driver model .....	60

# 1 Acronyms and definitions

Table 1: Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
ANSI	American National Standards Institute
API	Application Programming Interface
BSP	Board Support Package
COMP	Comparator
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRYP	Cryptographic processor unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DMA	Direct Memory Access
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LCD	Liquid Crystal Display Controller
MSP	MCU Specific Package
NVIC	Nested Vectored Interrupt Controller
PCD	USB Peripheral Controller Driver
PWR	Power controller
RCC	Reset and clock controller
RNG	Random Number Generator
RTC	Real-time clock
SD	Secure Digital
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
TSC	Touch Sensing Controller

<b>Acronym</b>	<b>Definition</b>
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

## 2 Overview of HAL drivers

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

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

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

The HAL main features are the following:

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

### 2.1 HAL and user-application files

#### 2.1.1 HAL driver files

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

**Table 2: HAL drivers files**

File	Description
<i>stm32f0xx_hal_ppp.c</i>	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. <i>Example: stm32f0xx_hal_adc.c, stm32f0xx_hal_irda.c, ...</i>
<i>stm32f0xx_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: stm32f0xx_hal_adc.h, stm32f0xx_hal_irda.h, ...</i>

File	Description
<i>stm32f0xx_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: stm32f0xx_hal_adc_ex.c, stm32f0xx_hal_dma_ex.c, ...</i>
<i>stm32f0xx_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: stm32f0xx_hal_adc_ex.h, stm32f0xx_hal_dma_ex.h, ...</i>
<i>stm32f0xx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
<i>stm32f0xx_hal.h</i>	<i>stm32f0xx_hal.c</i> header file
<i>stm32f0xx_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>stm32f0xx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application.
<i>stm32f0xx_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_stm32f0xx.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_stm32f0xx.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>stm32f0xx_flash.icf (optional)</i>	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
<i>stm32f0xx_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>stm32f0xx_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>stm32f0xx_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>stm32f0xx_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.

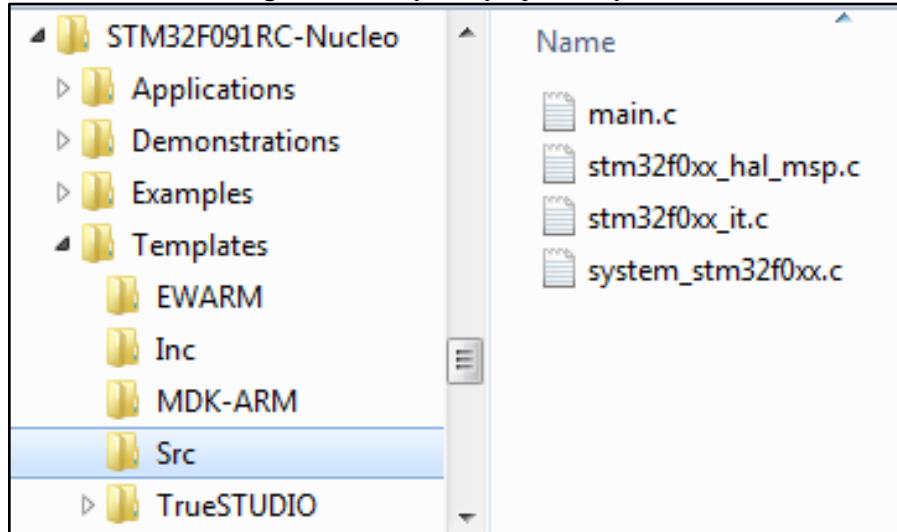
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Their characteristics are the following:

- 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(STM32F042xx) || defined(STM32F048xx) || defined(STM32F072xB) ||
defined(STM32F078xx) || \
defined(STM32F091xC) || defined(STM32F098xx)
#endif /* STM32F042xx || STM32F048xx || STM32F072xB || STM32F078xx || */
/* STM32F091xC || STM32F098xx */
```

The data structure related to the specific APIs is delimited by the device part number define statement. It is located in the corresponding extension header C file.

The following table summarizes the location of the different categories of HAL APIs in the driver files.

Table 4: APIs classification

	Generic file	Extension file
Common APIs	X	X <sup>(1)</sup>
Family specific APIs		X
Device specific APIs		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

IP/Module	STM32F030x6	STM32F030x8	STM32F070x6	STM32F070x8	STM32F030xC	STM32F031x6	STM32F051x8	STM32F071xB	STM32F091xC	STM32F042x6	STM32F072xB	STM32F048xx	STM32F058xx	STM32F078xx	STM32F098xx
stm32f0xx_hal.c	Yes														
stm32f0xx_hal_adc.c	Yes														
stm32f0xx_hal_adc_ex.c	Yes														
stm32f0xx_hal_can.c	No	Yes	Yes	Yes	Yes	No	Yes	Yes							
stm32f0xx_hal_cec.c	No	No	No	No	No	No	Yes								
stm32f0xx_hal_comp.c	No	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32f0xx_hal_cortex.c	Yes														
stm32f0xx_hal_crc.c	Yes														
stm32f0xx_hal_crc_ex.c	Yes														
stm32f0xx_hal_dac.c	No	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32f0xx_hal_dac_ex.c	No	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32f0xx_hal_dma.c	Yes														
stm32f0xx_hal_flash.c	Yes														

## Overview of HAL drivers

**UM1785**

IP/Module	STM32F030x6	STM32F030x8	STM32F070x6	STM32F070x8	STM32F030xC	STM32F031x6	STM32F051x8	STM32F071xB	STM32F091xC	STM32F042x6	STM32F072xB	STM32F048xx	STM32F058xx	STM32F078xx	STM32F098xx
stm32f0xx_hal_flash_ex.c	Yes														
stm32f0xx_hal_gpio.c	Yes														
stm32f0xx_hal_i2c.c	Yes														
stm32f0xx_hal_i2c_ex.c	Yes														
stm32f0xx_hal_i2s.c	No	No	No	No	No	Yes									
stm32f0xx_hal_irda.c	No	No	No	No	No	Yes									
stm32f0xx_hal_iwdg.c	Yes														
stm32f0xx_hal_msp_template.c	NA														
stm32f0xx_hal_pcd.c	No	No	Yes	Yes	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No
stm32f0xx_hal_pcd_ex.c	No	No	Yes	Yes	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No
stm32f0xx_hal_pwr.c	Yes														
stm32f0xx_hal_pwr_ex.c	Yes	No	No	No											
stm32f0xx_hal_rcc.c	Yes														
stm32f0xx_hal_rcc_ex.c	Yes														
stm32f0xx_hal_rtc.c	Yes														
stm32f0xx_hal_rtc_ex.c	Yes														
stm32f0xx_hal_smartcard.c	No	No	No	No	No	Yes									
stm32f0xx_hal_smartcard_ex.c	No	No	No	No	No	Yes									
stm32f0xx_hal_smbus.c	Yes														
stm32f0xx_hal_spi.c	Yes														
stm32f0xx_hal_tim.c	Yes														
stm32f0xx_hal_tim_ex.c	Yes														
stm32f0xx_hal_tsc.c	No	No	No	No	No	No	Yes								
stm32f0xx_hal_uart.c	Yes														
stm32f0xx_hal_uart_ex.c	Yes														
stm32f0xx_hal_usart.c	Yes														
stm32f0xx_hal_wwdg.c	Yes														

## 2.5 HAL drivers rules

### 2.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

**Table 6: HAL API naming rules**

	Generic	Family specific	Device specific
<b>File names</b>	<code>stm32f0xx_hal_ppp (c/h)</code>	<code>stm32f0xx_hal_ppp_ex (c/h)</code>	<code>stm32f0xx_hal_ppp_ex (c/h)</code>
<b>Module name</b>	<code>HAL_PPP_MODULE</code>		
<b>Function name</b>	<code>HAL_PPP_Function</code> <code>HAL_PPP_FeatureFunction_MODE</code>	<code>HAL_PPPEX_Function</code> <code>HAL_PPPEX_FeatureFunction_MODE</code>	<code>HAL_PPPEX_Function</code> <code>HAL_PPPEX_FeatureFunction_MODE</code>
<b>Handle name</b>	<code>PPP_HandleTypeDef</code>	NA	NA
<b>Init structure name</b>	<code>PPP_InitTypeDef</code>	NA	<code>PPP_InitTypeDef</code>
<b>Enum name</b>	<code>HAL_PPP_StructnameTypeDef</code>	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 STM32F0xx reference manuals.
- Peripheral registers are declared in the PPP\_TypeDef structure (e.g. ADC\_TypeDef) in the CMSIS header file corresponding to the selected platform: stm32f030x6.h, stm32f030x8.h, stm32f031x6.h, stm32f038xx.h, stm32f042x6.h, stm32f048xx.h, stm32f051x8.h, stm32f058xx.h, stm32f071xB.h, stm32f072xB.h, stm32f078xx.h, stm32f091xC.h and stm32f098xx.h. The platform is selected by enabling the compilation switch in the compilation toolchain directive or in the stm32f0xx.h file.
- 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\_Delnit, e.g. TIM\_Delnit.
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL\_PPP\_Function\_DMA ()*.
- The **Feature** prefix should refer to the new feature. Example: *HAL\_ADC\_Start()* refers to the injection mode

## 2.5.2 HAL general naming rules

- For the shared and system peripherals, no handle or instance object is used. This rule applies to the following peripherals:
  - GPIO
  - SYSTICK
  - NVIC
  - RCC
  - FLASH.

Example: The *HAL\_GPIO\_Init()* requires only the GPIO address and its configuration parameters.

```
HAL_StatusTypeDef HAL_GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef *Init)
{
/*GPIO Initialization body */
}
```

- The macros that handle interrupts and specific clock configurations are defined in each peripheral/module driver. These macros are exported in the peripheral driver header files so that they can be used by the extension file. The list of these macros is defined below: This list is not exhaustive and other macros related to peripheral features can be added, so that they can be used in the user application.

Table 7: Macros handling interrupts and specific clock configurations

Macros	Description
<code>_HAL_PPP_ENABLE_IT(_HANDLE_, _INTERRUPT_)</code>	Enables a specific peripheral interrupt
<code>_HAL_PPP_DISABLE_IT(_HANDLE_, _INTERRUPT_)</code>	Disables a specific peripheral interrupt
<code>_HAL_PPP_GET_IT (_HANDLE_, _INTERRUPT_)</code>	Gets a specific peripheral interrupt status
<code>_HAL_PPP_CLEAR_IT (_HANDLE_, _INTERRUPT_)</code>	Clears a specific peripheral interrupt status
<code>_HAL_PPP_GET_FLAG (_HANDLE_, _FLAG_)</code>	Gets a specific peripheral flag status
<code>_HAL_PPP_CLEAR_FLAG (_HANDLE_, _FLAG_)</code>	Clears a specific peripheral flag status
<code>_HAL_PPP_ENABLE(_HANDLE_)</code>	Enables a peripheral
<code>_HAL_PPP_DISABLE(_HANDLE_)</code>	Disables a peripheral
<code>_HAL_PPP_XXXX (_HANDLE_, _PARAM_)</code>	Specific PPP HAL driver macro
<code>_HAL_PPP_GET_IT_SOURCE (_HANDLE_, _INTERRUPT_)</code>	Checks the source of specified interrupt

- NVIC and SYSTICK are two ARM Cortex core features. The APIs related to these features are located in the `stm32f0xx_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 `stm32f0xx_it.c`
- User callback functions.

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

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: `HAL_PPP_MspInit()` and `HAL_PPP_MspDeInit()`
- Process complete callbacks : `HAL_PPP_ProcessCpltCallback`
- Error callback: `HAL_PPP_ErrorCallback`.

**Table 8: Callback functions**

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

## 2.6 HAL generic APIs

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

- **Initialization and de-initialization functions:** HAL\_PPP\_Init(), HAL\_PPP\_DeInit()
- **IO operation functions:** HAL\_PPP\_Read(), HAL\_PPP\_Write(), HAL\_PPP\_Transmit(), HAL\_PPP\_Receive()
- **Control functions:** HAL\_PPP\_Set(), HAL\_PPP\_Get().
- **State and Errors functions:** HAL\_PPP\_GetState(), HAL\_PPP\_GetError().

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

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

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

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

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

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

**Table 9: HAL generic APIs**

Function Group	Common API Name	Description
<i>Initialization group</i>	<i>HAL_ADC_Init()</i>	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	<i>HAL_ADC_DeInit()</i>	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
<i>IO operation group</i>	<i>HAL_ADC_Start()</i>	This function starts ADC conversions when the polling method is used
	<i>HAL_ADC_Stop()</i>	This function stops ADC conversions when the polling method is used
	<i>HAL_ADC_PollForConversion()</i>	This function allows waiting for the end of conversions when the polling method is used. In this case, a timeout value is specified by the user according to the application.
	<i>HAL_ADC_Start_IT()</i>	This function starts ADC conversions when the interrupt method is used
	<i>HAL_ADC_Stop_IT()</i>	This function stops ADC conversions when the interrupt method is used
	<i>HAL_ADC_IRQHandler()</i>	This function handles ADC interrupt requests

Function Group	Common API Name	Description
<i>Control group</i>	<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, `stm32f0xx_hal_ppp_ex.c`, that includes all the specific functions and define statements (`stm32f0xx_hal_ppp_ex.h`) for a given part number.

Below an example based on the ADC peripheral:

Table 10: HAL extension APIs

Function Group	Common API Name
<i>HAL_ADCEx_CalibrationStart()</i>	This function is used to start the automatic ADC calibration
<i>HAL_ADCEx_Calibration_GetValue()</i>	This function is used to get the ADC calibration factor
<i>HAL_ADCEx_Calibration_SetValue()</i>	This function is used to set the calibration factor to overwrite automatic conversion result

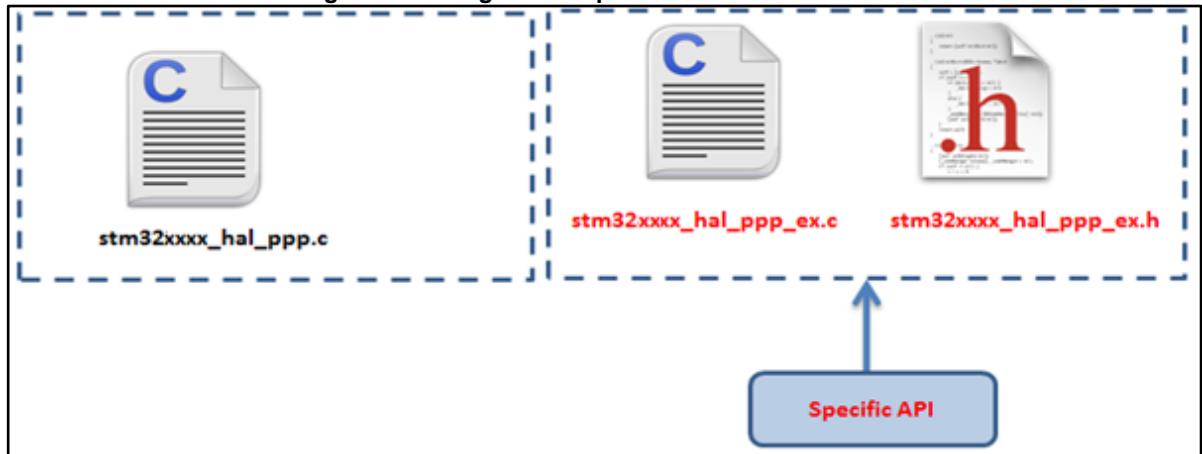
### 2.7.2 HAL extension model cases

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

#### Case1: Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the `stm32f0xx_hal_ppp_ex.c` extension file. They are named `HAL_PPPEX_Function()`.

Figure 2: Adding device-specific functions



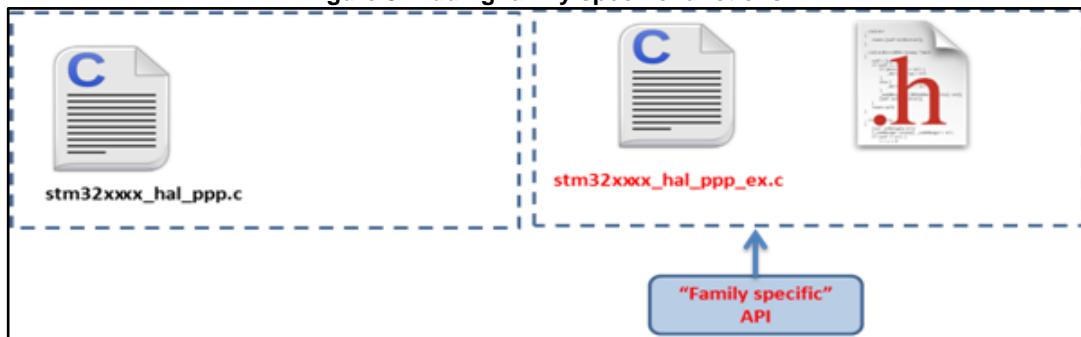
Example: `stm32f0xx_hal_rcc_ex.c/h`

```
#if defined(STM32F042xB) || defined(STM32F048xx) || \
defined(STM32F071xB) || defined(STM32F072xB) || defined(STM32F078xx) || \
defined(STM32F091xC) || defined(STM32F098xx)
void HAL_RCCEx_CRSConfig(RCC_CRSInitTypeDef *pInit);
void HAL_RCCEx_CRSSoftwareSynchronizationGenerate(void);
void HAL_RCCEx_CRSGetSynchronizationInfo(RCC_CRSSynchroInfoTypeDef *pSynchroInfo);
RCC_CRSStatusTypeDef HAL_RCCEx_CRSWaitSynchronization(uint32_t Timeout);
#endif /* STM32F042xB || STM32F048xx || */
/* STM32F071xB || STM32F072xB || STM32F078xx || */
/* STM32F091xC || STM32F098xx */
```

### Case2: Adding a family-specific function

In this case, the API is added in the extension driver C file and named `HAL_PPPEEx_Function()`.

Figure 3: Adding family-specific functions

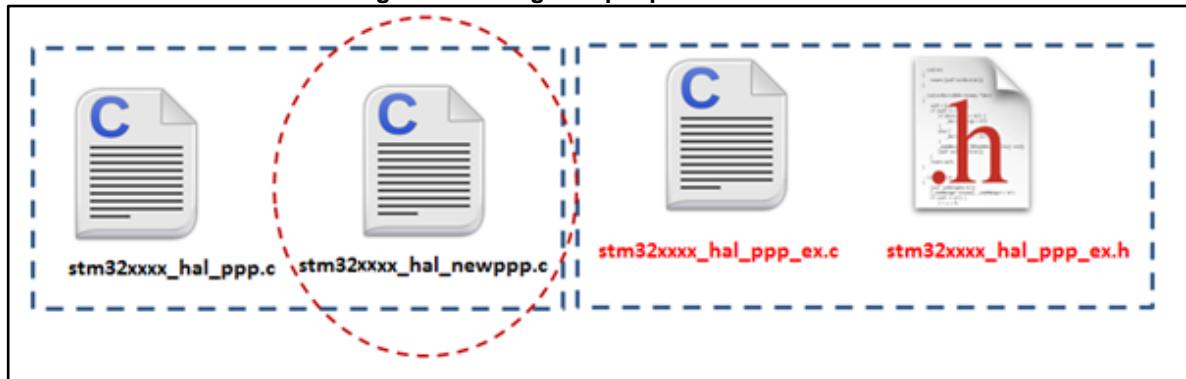


### Case3 : Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in `stm32f0xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32lx_hal_conf.h` using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

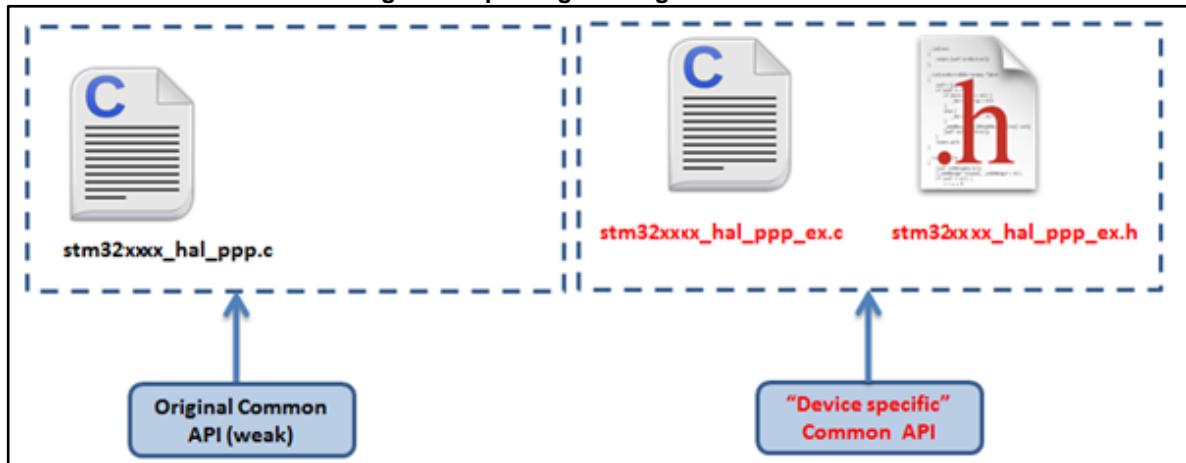


Example: stm32f0xx\_hal\_lcd.c/h

#### Case4: Updating existing common APIs

In this case, the routines are defined with the same names in the `stm32f0xx_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:

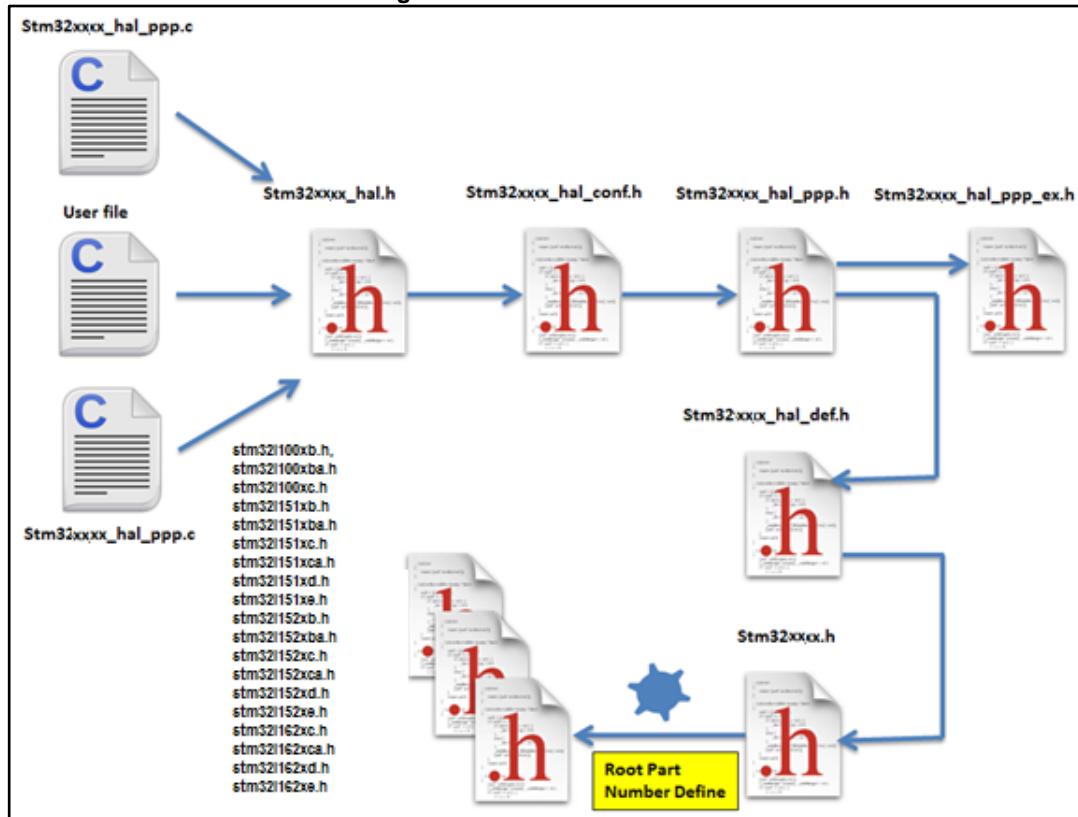
```
#if defined (STM32F072xB)
typedef struct
{
(...)

}PPP_InitTypeDef;
#endif /* STM32F072xB */
```

## 2.8 File inclusion model

The header of the common HAL driver file (`stm32f0xx_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 stm32f0xx_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 *stm32f0xx\_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 stm32f0xx_hal_def.h file
calls the stm32f0xx.h file in CMSIS library to get the data structures and the
address mapping for all peripherals:
```

- Declarations of peripheral registers and bits definition.
- Macros to access peripheral registers hardware (Write register, Read register...etc.).

- **Common macros**

- Macros defining NULL and HAL\_MAX\_DELAY

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

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

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

## 2.10 HAL configuration

The configuration file, *stm32f0xx\_hal\_conf.h*, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

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

**Table 11: Define statements used for HAL configuration**

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	8 000 000 (Hz)
HSE_STARTUP_TIMEOUT	Timeout for HSE start up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)
HSI_STARTUP_TIMEOUT	Timeout for HSI start up, expressed in ms	5000
LSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 (Hz)
HSI14_VALUE	Defines the value of the Internal High Speed oscillator for ADC expressed in Hz. The real value may vary depending on the variations in voltage and temperature.	14 000 000 (Hz)

Configuration item	Description	Default Value
<b>HSI48_VALUE</b>	Defines the value of the Internal High Speed oscillator for USB expressed in Hz. The real value may vary depending on the variations in voltage and temperature.	48 000 000 (Hz)
<b>LSI_VALUE</b>	Defines the value of the Internal Low Speed oscillator expressed in Hz. The real value may vary depending on the variations in voltage and temperature.	40 000 (Hz)
<b>VDD_VALUE</b>	VDD value	3300 (mV)
<b>USE_RTOS</b>	Enables the use of RTOS	FALSE (for future use)
<b>PREFETCH_ENABLE</b>	Enables prefetch feature	TRUE



The `stm32f0xx_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 `stm32f0xx_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 and APB 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 `stm32f0xx_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, ...)
- MCO and CSS configuration functions

A set of macros are defined in `stm32f0xx_hal_rcc.h` and `stm32f0xx_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 `stm32f0xx_it.c` and implement `HAL_GPIO_EXTI_Callback()`

The table below describes the `GPIO_InitTypeDef` structure field.

**Table 12: Description of `GPIO_InitTypeDef` structure**

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: <code>GPIO_PIN_x</code> or <code>GPIO_PIN_All</code> , where <code>x[0..15]</code>
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: <ul style="list-style-type: none"> <li>• <u>GPIO mode</u> <ul style="list-style-type: none"> <li>– <code>GPIO_MODE_INPUT</code> : Input Floating</li> <li>– <code>GPIO_MODE_OUTPUT_PP</code> : Output Push Pull</li> <li>– <code>GPIO_MODE_OUTPUT_OD</code> : Output Open Drain</li> <li>– <code>GPIO_MODE_AF_PP</code> : Alternate Function Push Pull</li> <li>– <code>GPIO_MODE_AF_OD</code> : Alternate Function Open Drain</li> <li>– <code>GPIO_MODE_ANALOG</code> : Analog mode</li> </ul> </li> <li>• <u>External Interrupt Mode</u> <ul style="list-style-type: none"> <li>– <code>GPIO_MODE_IT_RISING</code> : Rising edge trigger detection</li> <li>– <code>GPIO_MODE_IT_FALLING</code> : Falling edge trigger detection</li> <li>– <code>GPIO_MODE_IT_RISING_FALLING</code> : Rising/Falling edge trigger detection</li> </ul> </li> <li>• <u>External Event Mode</u> <ul style="list-style-type: none"> <li>– <code>GPIO_MODE_EVT_RISING</code> : Rising edge trigger detection</li> <li>– <code>GPIO_MODE_EVT_FALLING</code> : Falling edge trigger detection</li> <li>– <code>GPIO_MODE_EVT_RISING_FALLING</code> : Rising/Falling edge trigger detection</li> </ul> </li> </ul>

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

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external 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);
```

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

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

### 2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, `stm32f0xx_hal_cortex.c`, provides APIs to handle NVIC and Systick. The supported APIs include:

- HAL\_NVIC\_SetPriority()

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

## 2.11.4 PWR

The PWR HAL driver handles power management. The features shared between all STM32 Series are listed below:

- PVD configuration, enabling/disabling and interrupt handling
  - HAL\_PWR\_PVDCfg()
  - 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()
- Backup domain configuration
  - HAL\_PWR\_EnableBkUpAccess() / HAL\_PWR\_DisableBkUpAccess()

## 2.11.5 EXTI

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

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

The EXTI lines connected internally to the PVD, RTC, USB, and COMP are configured within the HAL drivers of these peripheral through the macros given in the table below. The EXTI internal connections depend on the targeted STM32 microcontroller (refer to the product datasheet for more details):

**Table 13: Description of EXTI configuration macros**

Macros	Description
PPP_EXTI_LINE_FUNCTION	Defines the EXTI line connected to the internal peripheral. Example: <code>#define PWR_EXTI_LINE_PVD ((uint32_t)0x00010000) /*!&lt;External interrupt line 16 Connected to the PVD EXTI Line */</code>
_HAL_PPP_EXTI_ENABLE_IT	Enables a given EXTI line Example: <code>_HAL_PWR_PVD_EXTI_ENABLE_IT()</code>

Macros	Description
<code>_HAL_PPP_EXTI_DISABLE_IT</code>	Disables a given EXTI line. Example: <code>_HAL_PWR_PVD_EXTI_DISABLE_IT()</code>
<code>_HAL_PPP_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_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_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_EXTI_ENABLE_EVENT</code>	Enables event on a given EXTI Line Example: <code>_HAL_PWR_PVD_EXTI_ENABLE_EVENT()</code>
<code>_HAL_PPP_EXTI_DISABLE_EVENT</code>	Disables event on a given EXTI line Example: <code>_HAL_PWR_PVD_EXTI_DISABLE_EVENT()</code>

If the EXTI interrupt mode is selected, the user application must call `HAL_PPP_FUNCTION_IRQHandler()` (for example `HAL_PWR_PVD_IRQHandler()`), from `stm32f0xx_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
- FIFO mode and its Threshold (if needed)
- Burst mode for Source and/or Destination (if needed).

Two operating modes are available:

- Polling mode I/O operation
  - a. Use `HAL_DMA_Start()` to start DMA transfer when the source and destination addresses and the Length of data to be transferred have been configured.
  - b. Use `HAL_DMA_PollForTransfer()` to poll for the end of current transfer. In this case a fixed timeout can be configured depending on the user application.

- Interrupt mode I/O operation
  - a. Configure the DMA interrupt priority using HAL\_NVIC\_SetPriority()
  - b. Enable the DMA IRQ handler using HAL\_NVIC\_EnableIRQ()
  - c. Use HAL\_DMA\_Start\_IT() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been configured. In this case the DMA interrupt is configured.
  - d. Use HAL\_DMA\_IRQHandler() called under DMA\_IRQHandler() Interrupt subroutine
  - e. When data transfer is complete, HAL\_DMA\_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (i.e. a member of DMA handle structure).

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

- Use HAL\_DMA\_GetState() function to return the DMA state and HAL\_DMA\_GetError() in case of error detection.
- Use HAL\_DMA\_Abort() function to abort the current transfer

The most used DMA HAL driver macros are the following:

- \_\_HAL\_DMA\_ENABLE: enables the specified DMA Channels.
- \_\_HAL\_DMA\_DISABLE: disables the specified DMA Channels.
- \_\_HAL\_DMA\_GET\_FLAG: gets the DMA Channels pending flags.
- \_\_HAL\_DMA\_CLEAR\_FLAG: clears the DMA Channels pending flags.
- \_\_HAL\_DMA\_ENABLE\_IT: enables the specified DMA Channels interrupts.
- \_\_HAL\_DMA\_DISABLE\_IT: disables the specified DMA Channels interrupts.
- \_\_HAL\_DMA\_GET\_IT\_SOURCE: checks whether the specified DMA channel 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").



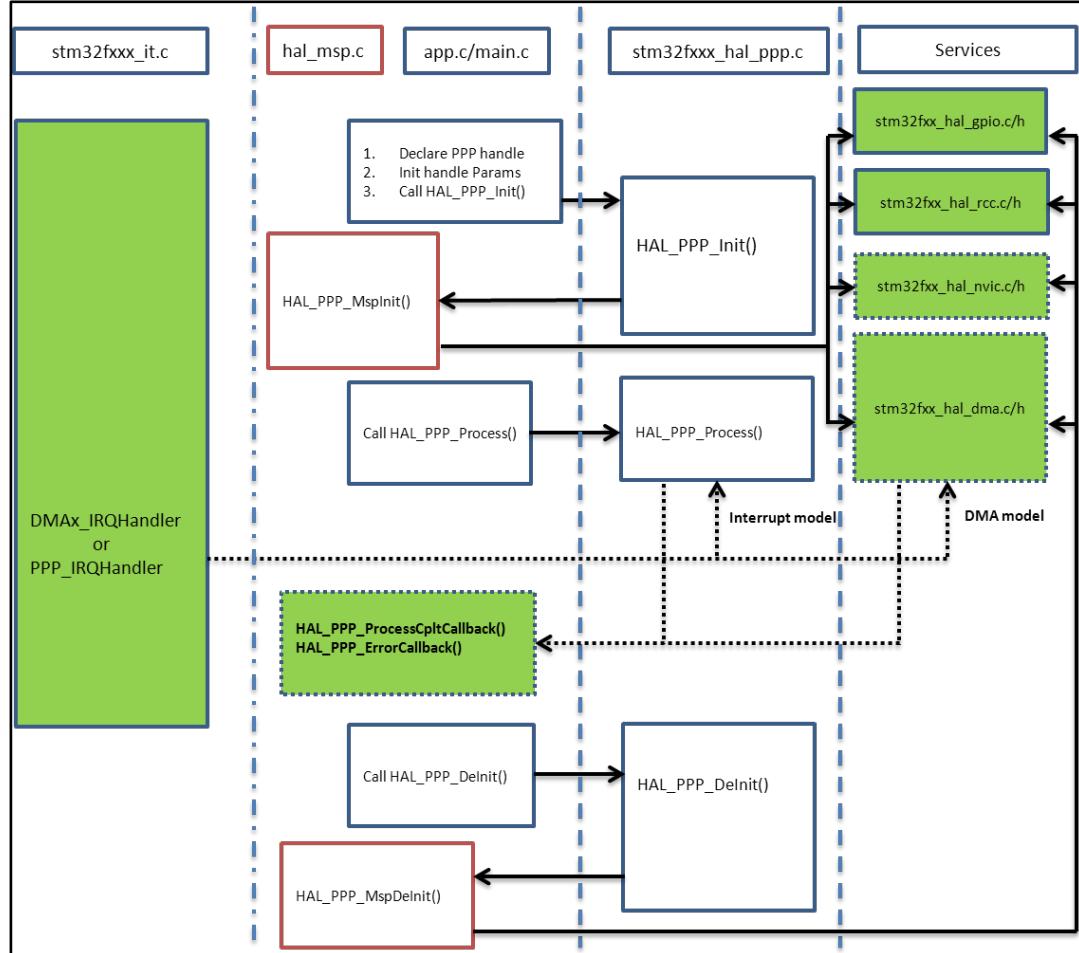
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

## 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**



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 `stm32f0xx_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\_DelInit()
  - Resets all peripherals
  - Calls function HAL\_MspDelInit() which a is user callback function to do system level De-Initializations.
- HAL\_GetTick(): this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- HAL\_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.  
Care must be taken when using HAL\_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL\_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be blocked.

### 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:

```
static void SystemClock_Config(void)
{
RCC_ClkInitTypeDef RCC_ClkInitStruct;
RCC_OscInitTypeDef RCC_OscInitStruct;
/* Enable HSE Oscillator and Activate PLL with HSE as source */
RCC_OscInitStruct.OscillatorType = RCC OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL6;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
Error_Handler();
}
/* Select PLL as system clock source and configure the HCLK, PCLK1 clocks dividers */
RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_PCLK1);
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
{
Error_Handler();
}
}
```

### 2.12.2.3 HAL MSP initialization process

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

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

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

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

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

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

The HAL MSP callback is implemented inside the *stm32f0xx\_hal\_msp.c* file in the user folders. An *stm32f0xx\_hal\_msp\_template.c* file 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.

*stm32f0xx\_hal\_msp.c* file contains the following functions:

Table 14: MSP functions

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

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

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

If there is nothing to be initialized by the global *HAL\_MspInit()* and *HAL\_MspDeInit()*, the two routines can simply be omitted.

### 2.12.3 HAL IO operation process

The HAL functions with internal data processing like Transmit, Receive, Write and Read are generally provided with three data processing modes as follows:

- Polling mode
- Interrupt mode

- DMA mode

### 2.12.3.1 Polling mode

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

The example below shows the typical polling mode processing sequence :

```
HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t pData,
int16_t Size, uint32_t Timeout)
{
if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(...) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(...)
return HAL_OK; }
```

### 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 *stm32f0xx\_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)
{
}

```

*stm32f0xx\_it.cfile:*

```

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 *stm32f0xx\_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 = USART1;
HAL_UART_Init(&UartHandle);
...
}

```

```

void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
  static DMA_HandleTypeDef hdma_tx;
  static DMA_HandleTypeDef hdma_rx;
  (...)

  __HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
  __HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
  (...)

}

```

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

An example of use is illustrated below:

*main.c* file:

```

UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User 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_ErrorCallback(UART_HandleTypeDef *phuart)
{
}

```

*stm32f0xx\_it.c* file:

```

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

```

`HAL_USART_TxCpltCallback()` and `HAL_USART_ErrorCallback()` should be linked in the `HAL_PPP_Process_DMA()` function to the DMA transfer complete callback and the DMA transfer Error callback by using the following statement:

```

HAL_PPP_Process_DMA (PPP_HandleTypeDef *hppp, Params....)
{
(...)

hppp->DMA_Handle->XferCpltCallback = HAL_UART_TxCpltCallback ;
hppp->DMA_Handle->XferErrorCallback = HAL_UART_ErrorCallback ;
(...)

}

```

## 2.12.4 Timeout and error management

### 2.12.4.1 Timeout management

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

```

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

```

The timeout possible value are the following:

**Table 15: Timeout values**

Timeout value	Description
0	No poll : Immediate process check and exit
1 ... (HAL_MAX_DELAY -1) <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 stm32fxxx\_hal\_def.h as 0xFFFFFFFF

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

Example of fixed timeout:

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

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

```
HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
    ...
    timeout = HAL_GetTick() + Timeout;
    ...
    while(ProcessOngoing)
    {
        ...
        if(Timeout != HAL_MAX_DELAY)
        {
            if(HAL_GetTick() >= timeout)
            {
                /* Process unlocked */
                __HAL_UNLOCK(hppp);
                hppp->State= HAL_PPP_STATE_TIMEOUT;
                return hppp->State;
            }
        }
        ...
    }
}
```

## 2.12.4.2 Error management

The HAL drivers implement a check for the following items:

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

```
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pdata, uint32
Size)
{
if ((pData == NULL) || (Size == 0))
{
return HAL_ERROR;
}
```

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

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

- Timeout error: the following statement is used when a timeout error occurs:

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

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

```
HAL_PPP_ErrorTypeDef HAL_PPP_GetError (PPP_HandleTypeDef *hppp);
```

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

```
typedef struct
{
PPP_TypeDef * Instance; /* PPP registers base address */
PPP_InitTypeDef Init; /* PPP initialization parameters */
HAL_LockTypeDef Lock; /* PPP locking object */
__IO HAL_PPP_StateTypeDef State; /* PPP state */
__IO HAL_PPP_ErrorTypeDef ErrorCode; /* PPP Error code */
(...)

/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

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

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

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

```
void HAL PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
    ErrorCode = HAL PPP_GetError (happ); /* retreive error code */
}
```

### 2.12.4.3 Run-time checking

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

To enable the run-time checking, use the `assert_param` macro, and leave the define `USE_FULL_ASSERT` uncommented in `stm32f0xx_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 `stm32f0xx_hal_conf.h`:

```
/* Exported macro -----*/
#ifndef USE_FULL_ASSERT
/**
 * @brief The assert_param macro is used for function's parameters check.
 * @param expr: If expr is false, it calls assert_failed function
 * which reports the name of the source file and the source
 * line number of the call that failed.
 * If expr is true, it returns no value.
 * @retval None */
#define assert_param(expr) ((expr)?(void)0:assert_failed((uint8_t *)__FILE__,
__LINE__))
/* Exported functions -----*/
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr)((void)0)
#endif /* USE_FULL_ASSERT */
```

The `assert_failed` function is implemented in the `main.c` file or in any other user C file:

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

```
/* Infinite loop */  
while (1)  
{  
}  
}
```



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

## 3 HAL 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:

- HAL Initialization and de-initialization functions
- HAL Control functions

#### 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\_SuspendTick()*
- *HAL\_ResumeTick()*
- *HAL\_Delay()*
- *HAL\_GetHalVersion()*
- *HAL\_GetREVID()*
- *HAL\_GetDEVID()*
- *HAL\_EnableDBGStopMode()*
- *HAL\_DisableDBGStopMode()*
- *HAL\_EnableDBGStandbyMode()*
- *HAL\_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.
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function is called at the beginning of program after reset and before the clock configuration</li> <li>• The time base configuration is based on HSI clock when exiting from Reset. Once done, time base tick start incrementing. In the default implementation, SysTick is used as source of time base. The tick variable is incremented each 1ms in its ISR.</li> </ul>

### 3.1.5 HAL\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_DelInit ( 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>• <b>HAL status</b></li> </ul>

## Notes

- This function is optional.

### 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>
Notes	<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>
Notes	<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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig().</li><li>• In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a</li></ul>

peripheral ISR process, The the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked. The function is declared as \_\_Weak to be overwritten in case of other implementation in user file.

### 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>
Notes	<ul style="list-style-type: none"><li>In the default implementation, this variable is incremented each 1ms in Systick ISR.</li><li>This function is declared as __weak to be overwritten in case of other implementations in user file.</li></ul>

### 3.1.10 HAL\_GetTick

Function Name	<b>uint32_t HAL_GetTick ( void )</b>
Function Description	Povides a tick value in millisecond.
Return values	<ul style="list-style-type: none"><li><b>tick value</b></li></ul>
Notes	<ul style="list-style-type: none"><li>The function is declared as __Weak to be overwritten in case of other implementations in user file.</li></ul>

### 3.1.11 HAL\_SuspendTick

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

---

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

### 3.1.12 HAL\_ResumeTick

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

### 3.1.13 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>
Notes	<ul style="list-style-type: none"><li>In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented.</li><li>This function is declared as __weak to be overwritten in case of other implementations in user file.</li></ul>

### 3.1.14 HAL\_GetHalVersion

Function Name	<code>uint32_t HAL_GetHalVersion ( void )</code>
Function Description	This method returns the HAL revision.
Return values	<ul style="list-style-type: none"><li>version : 0xXYZR (8bits for each decimal, R for RC)</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 3.1.15 HAL\_GetREVID

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

### 3.1.16 HAL\_GetDEVID

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

### 3.1.17 HAL\_EnableDBGStopMode

Function Name	<code>void HAL_EnableDBGStopMode ( void )</code>
Function Description	Enable the Debug Module during STOP mode.

---

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

### 3.1.18 HAL\_DisableDBGStopMode

Function Name	<b>void HAL_DisableDBGStopMode ( void )</b>
Function Description	Disable the Debug Module during STOP mode.
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 3.1.19 HAL\_EnableDBGStandbyMode

Function Name	<b>void HAL_EnableDBGStandbyMode ( void )</b>
Function Description	Enable the Debug Module during STANDBY mode.
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 3.1.20 HAL\_DisableDBGStandbyMode

Function Name	<b>void HAL_DisableDBGStandbyMode ( void )</b>
Function Description	Disable the Debug Module during STANDBY mode.
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 3.2 HAL Firmware driver defines

### 3.2.1 HAL

HAL

***HAL FastModePlus I2C***

IS\_HAL\_SYSCFG\_FASTMODEPLUS\_CONFIG

***HAL Fast mode plus driving cap***

\_HAL\_SYSCFG\_FASTMODEPLUS\_ENABLE

\_HAL\_SYSCFG\_FASTMODEPLUS\_DISABLE

***HAL IRDA Enveloppe Selection***

HAL\_SYSCFG\_IRDA\_ENV\_SEL\_TIM16

HAL\_SYSCFG\_IRDA\_ENV\_SEL\_USART1

HAL\_SYSCFG\_IRDA\_ENV\_SEL\_USART4

IS\_HAL\_SYSCFG\_IRDA\_ENV\_SEL

***HAL ISR Wrapper***

HAL\_SYSCFG\_ITLINE0

HAL\_SYSCFG\_ITLINE1

HAL\_SYSCFG\_ITLINE2

HAL\_SYSCFG\_ITLINE3

HAL\_SYSCFG\_ITLINE4

HAL\_SYSCFG\_ITLINE5

HAL\_SYSCFG\_ITLINE6

HAL\_SYSCFG\_ITLINE7

HAL\_SYSCFG\_ITLINE8

HAL\_SYSCFG\_ITLINE9

HAL\_SYSCFG\_ITLINE10

HAL\_SYSCFG\_ITLINE11

HAL\_SYSCFG\_ITLINE12

HAL\_SYSCFG\_ITLINE13

HAL\_SYSCFG\_ITLINE14

HAL\_SYSCFG\_ITLINE15

HAL\_SYSCFG\_ITLINE16

HAL\_SYSCFG\_ITLINE17

HAL\_SYSCFG\_ITLINE18

HAL\_SYSCFG\_ITLINE19

HAL\_SYSCFG\_ITLINE20

HAL\_SYSCFG\_ITLINE21

HAL\_SYSCFG\_ITLINE22  
HAL\_SYSCFG\_ITLINE23  
HAL\_SYSCFG\_ITLINE24  
HAL\_SYSCFG\_ITLINE25  
HAL\_SYSCFG\_ITLINE26  
HAL\_SYSCFG\_ITLINE27  
HAL\_SYSCFG\_ITLINE28  
HAL\_SYSCFG\_ITLINE29  
HAL\_SYSCFG\_ITLINE30  
HAL\_SYSCFG\_ITLINE31  
HAL\_ITLINE\_EWDG  
HAL\_ITLINE\_PVDOUT  
HAL\_ITLINE\_VDDIO2  
HAL\_ITLINE\_RTC\_WAKEUP  
HAL\_ITLINE\_RTC\_TSTAMP  
HAL\_ITLINE\_RTC\_ALRA  
HAL\_ITLINE\_FLASH\_ITF  
HAL\_ITLINE\_CRS  
HAL\_ITLINE\_CLK\_CTRL  
HAL\_ITLINE\_EXTI0  
HAL\_ITLINE\_EXTI1  
HAL\_ITLINE\_EXTI2  
HAL\_ITLINE\_EXTI3  
HAL\_ITLINE\_EXTI4  
HAL\_ITLINE\_EXTI5  
HAL\_ITLINE\_EXTI6  
HAL\_ITLINE\_EXTI7  
HAL\_ITLINE\_EXTI8  
HAL\_ITLINE\_EXTI9  
HAL\_ITLINE\_EXTI10  
HAL\_ITLINE\_EXTI11  
HAL\_ITLINE\_EXTI12  
HAL\_ITLINE\_EXTI13  
HAL\_ITLINE\_EXTI14  
HAL\_ITLINE\_EXTI15  
HAL\_ITLINE\_TSC\_EOA

HAL_ITLINE_TSC_MCE	
HAL_ITLINE_DMA1_CH1	
HAL_ITLINE_DMA1_CH2	
HAL_ITLINE_DMA1_CH3	
HAL_ITLINE_DMA2_CH1	
HAL_ITLINE_DMA2_CH2	
HAL_ITLINE_DMA1_CH4	
HAL_ITLINE_DMA1_CH5	
HAL_ITLINE_DMA1_CH6	
HAL_ITLINE_DMA1_CH7	
HAL_ITLINE_DMA2_CH3	
HAL_ITLINE_DMA2_CH4	
HAL_ITLINE_DMA2_CH5	
HAL_ITLINE_ADC	
HAL_ITLINE_COMP1	
HAL_ITLINE_COMP2	
HAL_ITLINE_TIM1_BRK	
HAL_ITLINE_TIM1_UPD	
HAL_ITLINE_TIM1_TRG	
HAL_ITLINE_TIM1_CCU	
HAL_ITLINE_TIM1_CC	
HAL_ITLINE_TIM2	
HAL_ITLINE_TIM3	
HAL_ITLINE_DAC	
HAL_ITLINE_TIM6	
HAL_ITLINE_TIM7	
HAL_ITLINE_TIM14	
HAL_ITLINE_TIM15	
HAL_ITLINE_TIM16	
HAL_ITLINE_TIM17	
HAL_ITLINE_I2C1	
HAL_ITLINE_I2C2	
HAL_ITLINE_SPI1	
HAL_ITLINE_SPI2	
HAL_ITLINE_USART1	USART1 GLB Interrupt -> exti[25]
HAL_ITLINE_USART2	USART2 GLB Interrupt -> exti[26]

HAL\_ITLINE\_USART3

HAL\_ITLINE\_USART4

HAL\_ITLINE\_USART5

HAL\_ITLINE\_USART6

HAL\_ITLINE\_USART7

HAL\_ITLINE\_USART8

HAL\_ITLINE\_CAN

HAL\_ITLINE\_CEC

***HAL ISR wrapper check***

`_HAL_GET_PENDING_IT`

***Constants***

`_STM32F0xx_HAL_VERSION_MAIN` [31:24] main version

`_STM32F0xx_HAL_VERSION_SUB1` [23:16] sub1 version

`_STM32F0xx_HAL_VERSION_SUB2` [15:8] sub2 version

`_STM32F0xx_HAL_VERSION_RC` [7:0] release candidate

`_STM32F0xx_HAL_VERSION`

`IDCODE_DEVID_MASK`

***HAL SYSCFG IRDA modulation envelope selection***

`_HAL_SYSCFG_IRDA_ENV_SELECTION`

`_HAL_SYSCFG_GET_IRDA_ENV_SELECTION`

## 4 HAL ADC Generic Driver

### 4.1 ADC Firmware driver registers structures

#### 4.1.1 ADC\_InitTypeDef

*ADC\_InitTypeDef* is defined in the `stm32f0xx_hal_adc.h`

##### Data Fields

- *uint32\_t ClockPrescaler*
- *uint32\_t Resolution*
- *uint32\_t DataAlign*
- *uint32\_t ScanConvMode*
- *uint32\_t EOCSelection*
- *uint32\_t LowPowerAutoWait*
- *uint32\_t LowPowerAutoPowerOff*
- *uint32\_t ContinuousConvMode*
- *uint32\_t DiscontinuousConvMode*
- *uint32\_t ExternalTrigConv*
- *uint32\_t ExternalTrigConvEdge*
- *uint32\_t DMAContinuousRequests*
- *uint32\_t Overrun*

##### Field Documentation

- ***uint32\_t ADC\_InitTypeDef::ClockPrescaler*** Select ADC clock source (synchronous clock derived from APB clock or asynchronous clock derived from ADC dedicated HSI RC oscillator 14MHz) and clock prescaler. This parameter can be a value of ***ADC\_ClockPrescaler*** Note: In case of usage of the ADC dedicated HSI RC oscillator, it must be preliminarily enabled at RCC top level. Note: This parameter can be modified only if the ADC is disabled
- ***uint32\_t ADC\_InitTypeDef::Resolution*** Configures the ADC resolution. This parameter can be a value of ***ADC\_Resolution***
- ***uint32\_t ADC\_InitTypeDef::DataAlign*** Specifies whether the ADC data alignment is left or right. This parameter can be a value of ***ADC\_Data\_align***
- ***uint32\_t ADC\_InitTypeDef::ScanConvMode*** Configures the sequencer of regular group. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. Sequencer is automatically enabled if several channels are set (sequencer cannot be disabled, as it can be the case on other STM32 devices): If only 1 channel is set: Conversion is performed in single mode. If several channels are set: Conversions are performed in sequence mode (ranks defined by each channel number: channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Scan direction can be set to forward (from channel 0 to channel 18) or backward (from channel 18 to channel 0). This parameter can be a value of ***ADC\_Scan\_mode***
- ***uint32\_t ADC\_InitTypeDef::EOCSelection*** Specifies what EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of conversion of each rank or complete sequence. This parameter can be a value of ***ADC\_EOCSelection***.
- ***uint32\_t ADC\_InitTypeDef::LowPowerAutoWait*** Selects the dynamic low power Auto Delay: new conversion start only when the previous conversion (for regular group) or previous sequence (for injected group) has been treated by user software.

This feature automatically adapts the speed of ADC to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: Do not use with interruption or DMA (`HAL_ADC_Start_IT()`, `HAL_ADC_Start_DMA()`) since they have to clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion with `HAL_ADC_Start()`, 2. Later on, when conversion data is needed: use `HAL_ADC_PollForConversion()` to ensure that conversion is completed and use `HAL_ADC_GetValue()` to retrieve conversion result and trig another conversion.

- **`uint32_t ADC_InitTypeDef::LowPowerAutoPowerOff`** Selects the auto-off mode: the ADC automatically powers-off after a conversion and automatically wakes-up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with automatic wait mode (parameter 'LowPowerAutoWait'). This parameter can be set to ENABLE or DISABLE. Note: If enabled, this feature also turns off the ADC dedicated 14 MHz RC oscillator (HSI14)
- **`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::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 Note: Number of discontinuous ranks increment is fixed to one-by-one.
- **`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. This parameter can be a value of [`ADC\_External\_trigger\_source-Regular`](#)
- **`uint32_t ADC_InitTypeDef::ExternalTrigConvEdge`** Selects the external trigger edge of regular group. If trigger is set to ADC\_SOFTWARE\_START, this parameter is discarded. This parameter can be a value of [`ADC\_External\_trigger\_edge-Regular`](#)
- **`uint32_t ADC_InitTypeDef::DMAContinuousRequests`** Specifies whether the DMA requests are performed in one shot mode (DMA transfer stop when number of conversions is reached) or in Continuous mode (DMA transfer unlimited, whatever number of conversions). Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached. This parameter can be set to ENABLE or DISABLE.
- **`uint32_t ADC_InitTypeDef::Overrun`** Select the behaviour in case of overrun: data preserved or overwritten This parameter has an effect on regular group only, including in DMA mode. This parameter can be a value of [`ADC\_Overrun`](#)

#### 4.1.2 ADC\_ChannelConfTypeDef

`ADC_ChannelConfTypeDef` is defined in the `stm32f0xx_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.
- ***uint32\_t ADC\_ChannelConfTypeDef::Rank*** Add or remove the channel from ADC regular group sequencer. On STM32F0 devices, rank is defined by each channel number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Despite the channel rank is fixed, this parameter allow an additional possibility: to remove the selected rank (selected channel) from sequencer. This parameter can be a value of [\*\*ADC\\_rank\*\*](#)
- ***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, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of [\*\*ADC\\_sampling\\_times\*\*](#) Caution: this setting impacts the entire regular group. Therefore, call of [\*\*HAL\\_ADC\\_ConfigChannel\(\)\*\*](#) to configure a channel can impact the configuration of other channels previously set. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS\_vrefint, TS\_vbat, TS\_temp (values rough order: 5us to 17us).

### 4.1.3 ADC\_AnalogWDGConfTypeDef

**ADC\_AnalogWDGConfTypeDef** is defined in the `stm32f0xx_hal_adc.h`

#### Data Fields

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

### Field Documentation

- ***uint32\_t ADC\_AnalogWDGConfTypeDef::WatchdogMode*** Configures the ADC analog watchdog mode: single/all/none channels. This parameter can be a value of [\*\*ADC\\_analog\\_watchdog\\_mode\*\*](#).
- ***uint32\_t ADC\_AnalogWDGConfTypeDef::Channel*** Selects which ADC channel to monitor by analog watchdog. This parameter has an effect only if parameter 'WatchdogMode' is configured on single channel. Only 1 channel can be monitored. This parameter can be a value of [\*\*ADC\\_channels\*\*](#).
- ***uint32\_t ADC\_AnalogWDGConfTypeDef::ITMode*** Specifies whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t ADC\_AnalogWDGConfTypeDef::HighThreshold*** Configures the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F respectively.
- ***uint32\_t ADC\_AnalogWDGConfTypeDef::LowThreshold*** Configures the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10,

8 or 6 bits), this parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F respectively.

#### 4.1.4 ADC\_HandleTypeDef

*ADC\_HandleTypeDef* is defined in the `stm32f0xx_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 specific features

1. 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
2. Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
3. Single and continuous conversion modes.
4. Scan mode for automatic conversion of channel 0 to channel 'n'.
5. Data alignment with in-built data coherency.
6. Programmable sampling time.
7. ADC conversion group Regular.
8. External trigger (timer or EXTI) with configurable polarity.
9. DMA request generation for transfer of conversions data of regular group.
10. ADC calibration
11. ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
12. ADC input range: from Vref minus (connected to Vssa) to Vref plus (connected to Vdda or to an external voltage reference).

## 4.2.2 How to use this driver

1. Enable the ADC interface
  - As prerequisite, into HAL\_ADC\_MspInit(), ADC clock must be configured at RCC top level: clock source and clock prescaler.
  - Two possible clock sources: synchronous clock derived from APB clock or asynchronous clock derived from ADC dedicated HSI RC oscillator 14MHz.
  - Example: \_\_ADC1\_CLK\_ENABLE(); (mandatory) HI14 enable or let under control of ADC: (optional) RCC\_OsclInitTypeDef RCC\_OsclInitStructure; RCC\_OsclInitStructure.OscillatorType = RCC OSCILLATORTYPE\_HSI14; RCC\_OsclInitStructure.HSI14CalibrationValue = RCC\_HSI14CALIBRATION\_DEFAULT; RCC\_OsclInitStructure.HSI14State = RCC\_HSI14\_ADC\_CONTROL; RCC\_OsclInitStructure.PLL... (optional if used for system clock) HAL\_RCC\_OscConfig(&RCC\_OsclInitStructure); Parameter "HSI14State" must be set either: - to "...HSI14State = RCC\_HSI14\_ADC\_CONTROL" to let the ADC control the HSI14 oscillator enable/disable (if not used to supply the main system clock): feature used if ADC mode LowPowerAutoPowerOff is enabled. - to "...HSI14State = RCC\_HSI14\_ON" to maintain the HSI14 oscillator always enabled: can be used to supply the main system clock.
2. ADC pins configuration
  - Enable the clock for the ADC GPIOs using the following function: \_\_GPIOx\_CLK\_ENABLE();
  - Configure these ADC pins in analog mode using HAL\_GPIO\_Init();
3. Configure the ADC parameters (conversion resolution, data alignment, continuous mode, ...) using the HAL\_ADC\_Init() function.
4. Activate the ADC peripheral using one of the start functions: HAL\_ADC\_Start(), HAL\_ADC\_Start\_IT(), HAL\_ADC\_Start\_DMA().

### Channels configuration to regular group

- To configure the ADC regular group features, use HAL\_ADC\_Init() and HAL\_ADC\_ConfigChannel() functions.
- To activate the continuous mode, use the HAL\_ADC\_Init() function.
- To read the ADC converted values, use the HAL\_ADC\_GetValue() function.

### DMA for regular configuration

- To enable the DMA mode for regular group, use the HAL\_ADC\_Start\_DMA() function.
- To enable the generation of DMA requests continuously at the end of the last DMA transfer, use the HAL\_ADC\_Init() function.

## 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\\_DelInit\(\)\*](#)
- [\*HAL\\_ADC\\_MspInit\(\)\*](#)
- [\*HAL\\_ADC\\_MspDelInit\(\)\*](#)

#### 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 ( <i>ADC_HandleTypeDef</i> * 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• As prerequisite, ADC clock must be configured at RCC top level depending on both possible clock sources: APB clock or HSI clock. 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 ( <i>ADC_HandleTypeDef</i> * 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common group are disabled. If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behaviour in case of reset of a single ADC while the other ADCs sharing the same</li> </ul>

common group is still running.

#### 4.2.9 HAL\_ADC\_MspInit

Function Name	<b>void HAL_ADC_MspInit ( <i>ADC_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 4.2.10 HAL\_ADC\_MspDeInit

Function Name	<b>void HAL_ADC_MspDeInit ( <i>ADC_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 4.2.11 HAL\_ADC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start ( <i>ADC_HandleTypeDef</i> * 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 4.2.12 HAL\_ADC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Stop ADC conversion of regular group, 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>• <b>HAL status</b>.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.13 HAL\_ADC\_PollForConversion

Function Name	<b>HAL_StatusTypeDef HAL_ADC_PollForConversion ( <i>ADC_HandleTypeDef</i> * hadc, uint32_t Timeout)</b>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.14 HAL\_ADC\_PollForEvent

Function Name	<b>HAL_StatusTypeDef HAL_ADC_PollForEvent ( <i>ADC_HandleTypeDef</i> * hadc, uint32_t EventType, uint32_t Timeout)</b>
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: <ul style="list-style-type: none"> <li>– <b>AWD_EVENT</b> ADC Analog watchdog event</li> </ul> </li> </ul>

Return values

- **OVR\_EVENT** ADC Overrun event
- **Timeout** : Timeout value in millisecond.

Notes

- **HAL status**
- None.

#### 4.2.15 HAL\_ADC\_Start\_IT

Function Name

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

Function Description

Enables ADC, starts conversion of regular group with interruption.

Parameters

- **hadc** : ADC handle

Return values

- **HAL status**

Notes

- None.

#### 4.2.16 HAL\_ADC\_Stop\_IT

Function Name

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

Function Description

Stop ADC conversion of regular group, disable interruption of end-of-conversion, disable ADC peripheral.

Parameters

- **hadc** : ADC handle

Return values

- **HAL status**.

Notes

- None.

#### 4.2.17 HAL\_ADC\_Start\_DMA

Function Name

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

<b>Length)</b>	
Function Description	Enables ADC, starts conversion of regular group and transfers result through DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : ADC handle</li> <li>• <b>pData</b> : The destination Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from ADC peripheral to memory.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.18 HAL\_ADC\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop_DMA ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Stop ADC conversion of regular group, disable ADC DMA transfer, 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>• <b>HAL status.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.19 HAL\_ADC\_GetValue

Function Name	<b>uint32_t HAL_ADC_GetValue ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Get ADC regular group conversion result.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Converted value</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.20 HAL\_ADC\_IRQHandler

Function Name	<b>void HAL_ADC_IRQHandler ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Handles ADC interrupt request.
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>• None.</li></ul>

#### 4.2.21 HAL\_ADC\_ConvCpltCallback

Function Name	<b>void HAL_ADC_ConvCpltCallback ( <i>ADC_HandleTypeDef</i> * hadc)</b>
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 ( <i>ADC_HandleTypeDef</i> * 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 ( <i>ADC_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.24 HAL\_ADC\_ErrorCallback

Function Name	<b>void HAL_ADC_ErrorCallback ( <i>ADC_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.25 HAL\_ADC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_ADC_ConfigChannel ( <i>ADC_HandleTypeDef</i> * hadc, <i>ADC_ChannelConfTypeDef</i> * 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In case of usage of internal measurement channels: VrefInt/Vbat/TempSensor. Sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device</li> </ul>

- datasheet for timings values, parameters TS\_vrefint, TS\_vbat, TS\_temp (values rough order: 5us to 17us). These internal paths can be disabled using function HAL\_ADC\_Delnit().
- 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 (</b> <b>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><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Possibility to update parameters on the fly: This function initializes the selected analog watchdog, following calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" 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_AnalogWDGConfTypeDef".</li> </ul>

#### 4.2.27 HAL\_ADC\_GetState

Function Name	<b>HAL_ADC_StateTypeDef HAL_ADC_GetState (</b> <b>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><b>HAL state</b></li> </ul>

## Notes

- None.

### 4.2.28 HAL\_ADC\_GetError

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

## 4.3 ADC Firmware driver defines

### 4.3.1 ADC

ADC

***ADC analog watchdog mode***

`ADC_ANALOGWATCHDOG_NONE`

`ADC_ANALOGWATCHDOG_SINGLE_REG`

`ADC_ANALOGWATCHDOG_ALL_REG`

`IS_ADC_ANALOG_WATCHDOG_MODE`

***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_18	
ADC_CHANNEL_TEMPSENSOR	
ADC_CHANNEL_VREFINT	
ADC_CHANNEL_VBAT	
IS_ADC_CHANNEL	
<b>ADC ClockPrescaler</b>	
ADC_CLOCK_ASYNC	ADC asynchronous clock derived from ADC dedicated HSI
ADC_CLOCK_SYNC_PCLK_DIV2	ADC synchronous clock derived from AHB clock divided by a prescaler of 2
ADC_CLOCK_SYNC_PCLK_DIV4	ADC synchronous clock derived from AHB clock divided by a prescaler of 4
ADC_CLOCKPRESCALER_PCLK_DIV2	
ADC_CLOCKPRESCALER_PCLK_DIV4	
IS_ADC_CLOCKPRESCALER	
<b>ADC Data_align</b>	
ADC_DATAALIGN_RIGHT	
ADC_DATAALIGN_LEFT	
IS_ADC_DATA_ALIGN	
<b>ADC EOCSelection</b>	
EOC_SINGLE_CONV	
EOC_SEQ_CONV	
EOC_SINGLE_SEQ_CONV	reserved for future use
IS_ADC_EOC_SELECTION	
<b>ADC Error Code</b>	
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
<b>ADC Event type</b>	

AWD_EVENT	ADC Analog watchdog 1 event
OVR_EVENT	ADC overrun event

IS\_ADC\_EVENT\_TYPE

**ADC Exported Macros**`_HAL_ADC_RESET_HANDLE_STATE`**Description:**

- Reset ADC handle state.

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

- None:

`_HAL_ADC_IS_ENABLED`**Description:**

- Verification of ADC state: enabled or disabled.

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

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

`_HAL_ADC_IS_SOFTWARE_START_REGULAR`**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)

`_HAL_ADC_IS_CONVERSION_ONGOING_REGULAR`**Description:**

- Check if no conversion on going on regular group.

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

- SET: (conversion is on going) or RESET (no

conversion is on going)

#### `_HAL_ADC_GET_RESOLUTION`

**Description:**

- Returns resolution bits in CFGR1 register: RES[1:0].

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

- None:

**Description:**

- Returns ADC sample time bits in SMPR register: SMP[2:0].

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

- None:

**Description:**

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

**Parameters:**

- `_HANDLE_`: ADC handle
- `_INTERRUPT_`: ADC interrupt source to check

**Return value:**

- State: of interruption (SET or RESET)

**Description:**

- Enable the ADC end of conversion interrupt.

**Parameters:**

- `_HANDLE_`: ADC handle
- `_INTERRUPT_`: ADC Interrupt

**Return value:**

- None:

**Description:**

- Disable the ADC end of conversion interrupt.

**Parameters:**

- `_HANDLE_`: ADC handle
- `_INTERRUPT_`: ADC Interrupt

**Return value:**

- None:

`_HAL_ADC_GET_FLAG`**Description:**

- Get the selected ADC's flag status.

**Parameters:**

- `_HANDLE_`: ADC handle
- `_FLAG_`: ADC flag

**Return value:**

- None:

`_HAL_ADC_CLEAR_FLAG`**Description:**

- Clear the ADC's pending flags.

**Parameters:**

- `_HANDLE_`: ADC handle
- `_FLAG_`: ADC flag

**Return value:**

- None:

`_HAL_ADC_CLEAR_ERRORCODE`**Description:**

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

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

- None:

`_HAL_ADC_CHSELR_CHANNEL`**Description:**

- Configure the channel number into channel selection register.

**Parameters:**

- `_CHANNEL_`: ADC

Channel

**Return value:**

- None:

***ADC Exported Macro internal HAL driver***

`_HAL_ADC_CFGR_AWDCH`

**Description:**

- Set the Analog Watchdog 1 channel.

**Parameters:**

- `_CHANNEL_`: channel to be monitored by Analog Watchdog 1.

**Return value:**

- None:

**Description:**

- Enable ADC discontinuous conversion mode for regular group.

**Parameters:**

- `_REG_DISCONTINUOUS_MODE_`: Regulat discontinuous mode.

**Return value:**

- None:

**Description:**

- Enable the ADC auto off mode.

**Parameters:**

- `_AUTOOFF_`: Auto off bit enable or disable.

**Return value:**

- None:

**Description:**

- Enable the ADC auto delay mode.

**Parameters:**

- `_AUTOWAIT_`: Auto delay bit enable or disable.

**Return value:**

- None:

**Description:**

- Enable ADC continuous

conversion mode.

**Parameters:**

- `_CONTINUOUS_MODE_`: Continuous mode.

**Return value:**

- None:

**Description:**

- Enable ADC overrun mode.

**Parameters:**

- `_OVERRUN_MODE_`: Overrun mode.

**Return value:**

- Overrun: bit setting to be programmed into CFGR register

**Description:**

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

**Parameters:**

- `_SCAN_MODE_`: Scan conversion mode.

**Return value:**

- None:

**Description:**

- Enable the ADC DMA continuous request.

**Parameters:**

- `_DMACONTREQ_MODE_`: DMA continuous request mode.

**Return value:**

- None:

**Description:**

- Configure the channel number into offset OFRx register.

**Parameters:**

- `_CHANNEL_`: ADC Channel

**Return value:**

- None:

`__HAL_ADC_TRX_HIGHTHRESHOLD`

**Description:**

- Configure the analog watchdog high threshold into register TR.

**Parameters:**

- `_Threshold_`: Threshold value

**Return value:**

- None:

`__HAL_ADC_ENABLE`

**Description:**

- Enable the ADC peripheral.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None:

`__HAL_ADC_ENABLING_CONDITIONS`

**Description:**

- Verification of hardware constraints before ADC can be enabled.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- SET: (ADC can be enabled) or RESET (ADC cannot be enabled)

`__HAL_ADC_DISABLE`

**Description:**

- Disable the ADC peripheral.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None:

`__HAL_ADC_DISABLING_CONDITIONS`

**Description:**

- Verification of hardware constraints before ADC can be disabled.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- SET: (ADC can be disabled) or RESET (ADC cannot be disabled)

---

`__HAL_ADC_AWD1THRESHOLD_SHIFT_RESOLUTION`

**Description:**

- Shift the AWD threshold in function of the selected ADC resolution.

**Parameters:**

- `__HANDLE__`: ADC handle
- `_Threshold_`: Value to be shifted

**Return value:**

- None:

***ADC External trigger edge Regular***

`ADC_EXTERNALTRIGCONVEDGE_NONE`  
`ADC_EXTERNALTRIGCONVEDGE_RISING`  
`ADC_EXTERNALTRIGCONVEDGE_FALLING`  
`ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING`  
`IS_ADC_EXTTRIG_EDGE`

***ADC External trigger source Regular***

`ADC_EXTERNALTRIGCONV_T1_TRGO`  
`ADC_EXTERNALTRIGCONV_T1_CC4`  
`ADC_EXTERNALTRIGCONV_T2_TRGO`  
`ADC_EXTERNALTRIGCONV_T3_TRGO`  
`ADC_EXTERNALTRIGCONV_T15_TRGO`  
`ADC_SOFTWARE_START`  
`IS_ADC_EXTTRIG`

***ADC flags definition***

<code>ADC_FLAG_AWD</code>	ADC Analog watchdog flag
<code>ADC_FLAG_OVR</code>	ADC overrun flag
<code>ADC_FLAG_EOS</code>	ADC End of Regular sequence of Conversions flag
<code>ADC_FLAG_EOC</code>	ADC End of Regular Conversion flag
<code>ADC_FLAG_EOSMP</code>	ADC End of Sampling flag
<code>ADC_FLAG_RDY</code>	ADC Ready flag
<code>ADC_FLAG_ALL</code>	
<code>ADC_FLAG_POSTCONV_ALL</code>	

***ADC Internal HAL driver Ext trig src Regular***

`ADC1_2_EXTERNALTRIG_T1_TRGO`  
`ADC1_2_EXTERNALTRIG_T1_CC4`  
`ADC1_2_EXTERNALTRIG_T2_TRGO`  
`ADC1_2_EXTERNALTRIG_T3_TRGO`

ADC1\_2\_EXTERNALTRIG\_T15\_TRGO

***ADC interrupts definition***

ADC_IT_AWD	ADC Analog watchdog interrupt source
ADC_IT_OVR	ADC overrun interrupt source
ADC_IT_EOS	ADC End of Regular sequence of Conversions interrupt source
ADC_IT_EOC	ADC End of Regular Conversion interrupt source
ADC_IT_EOSMP	ADC End of Sampling interrupt source
ADC_IT_RDY	ADC Ready interrupt source

***ADC Overrun***

OVR\_DATA\_OVERWRITTEN  
OVR\_DATA\_PRESERVED  
IS\_ADC\_OVERRUN

***ADC Private Constants***

ADC\_ENABLE\_TIMEOUT  
ADC\_DISABLE\_TIMEOUT  
ADC\_STOP\_CONVERSION\_TIMEOUT  
ADC\_TEMPSENSOR\_DELAY\_CPU\_CYCLES  
ADC\_STAB\_DELAY\_CPU\_CYCLES

***ADC range verification***

IS\_ADC\_RANGE

***ADC rank***

ADC\_RANK\_CHANNEL\_NUMBER      Enable the rank of the selected channels. Rank is defined by each channel number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...)  
ADC\_RANK\_NONE      Disable the selected rank (selected channel) from sequencer

IS\_ADC\_RANK

***ADC regular rank verification***

IS\_ADC\_REGULAR\_RANK

***ADC Resolution***

ADC\_RESOLUTION12b      ADC 12-bit resolution  
ADC\_RESOLUTION10b      ADC 10-bit resolution  
ADC\_RESOLUTION8b      ADC 8-bit resolution  
ADC\_RESOLUTION6b      ADC 6-bit resolution  
IS\_ADC\_RESOLUTION

***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
IS_ADC_SAMPLE_TIME	
<b>ADC Scan mode</b>	
ADC_SCAN_DIRECTION_FORWARD	Scan direction forward: from channel 0 to channel 18
ADC_SCAN_DIRECTION_BACKWARD	Scan direction backward: from channel 18 to channel 0
ADC_SCAN_ENABLE	
IS_ADC_SCAN_MODE	

## 5 HAL ADC Extension Driver

### 5.1 ADCEx Firmware driver API description

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

#### 5.1.1 ADC specific features

1. 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
2. Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
3. Single and continuous conversion modes.
4. Scan mode for automatic conversion of channel 0 to channel 'n'.
5. Data alignment with in-built data coherency.
6. Programmable sampling time.
7. ADC conversion group Regular.
8. External trigger (timer or EXTI) with configurable polarity.
9. DMA request generation for transfer of conversions data of regular group.
10. ADC calibration
11. ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
12. ADC input range: from Vref minud (connected to Vssa) to Vref plus(connected to Vdda or to an external voltage reference).

#### 5.1.2 How to use this driver

1. Enable the ADC interface As prerequisite, into HAL\_ADC\_MspInit(), ADC clock must be configured at RCC top level: clock source and clock prescaler. Two possible clock sources: synchronous clock derived from APB clock or asynchronous clock derived from ADC dedicated HSI RC oscillator 14MHz. Example: \_\_ADC1\_CLK\_ENABLE(); (mandatory) HSI14 enable or let under control of ADC: (optional) RCC\_OscInitTypeDef RCC\_OscInitStructure; RCC\_OscInitStructure.OscillatorType = RCC\_OSCILLATORTYPE\_HSI14; RCC\_OscInitStructure.HSI14CalibrationValue = RCC\_HSI14CALIBRATION\_DEFAULT; RCC\_OscInitStructure.HSI14State = RCC\_HSI14\_ADC\_CONTROL; RCC\_OscInitStructure.PLL... (optional if used for system clock) HAL\_RCC\_OscConfig(&RCC\_OscInitStructure); Parameter "HSI14State" must be set either: - to "...HSI14State = RCC\_HSI14\_ADC\_CONTROL" to let the ADC control the HSI14 oscillator enable/disable (if not used to supply the main system clock): feature used if ADC mode LowPowerAutoPowerOff is enabled. - to "...HSI14State = RCC\_HSI14\_ON" to maintain the HSI14 oscillator always enabled: can be used to supply the main system clock.
2. ADC pins configuration
  - Enable the clock for the ADC GPIOs using the following function:  
\_\_GPIOx\_CLK\_ENABLE();
  - Configure these ADC pins in analog mode using HAL\_GPIO\_Init();
3. Configure the ADC parameters (conversion resolution, data alignment, continuous mode, ...) using the HAL\_ADC\_Init() function.

4. Activate the ADC peripheral using one of the start functions: HAL\_ADC\_Start(), HAL\_ADC\_Start\_IT(), HAL\_ADC\_Start\_DMA().

### Regular channels group configuration

- To configure the ADC regular channels group features, use HAL\_ADC\_Init() and HAL\_ADC\_ConfigChannel() functions.
- To activate the continuous mode, use the HAL\_ADC\_Init() function.
- To read the ADC converted values, use the HAL\_ADC\_GetValue() function.

### DMA for Regular channels group features configuration

- To enable the DMA mode for regular channels group, use the HAL\_ADC\_Start\_DMA() function.
- To enable the generation of DMA requests continuously at the end of the last DMA transfer, use the HAL\_ADC\_Init() function.

## 5.1.3 IO operation functions

This section provides functions allowing to:

- Perform the ADC calibration.
- [\*\*HAL\\_ADCEx\\_Calibration\\_Start\(\)\*\*](#)

## 5.1.4 HAL\_ADCEx\_Calibration\_Start

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (</b> <a href="#"><b>ADC_HandleTypeDef * hadc</b></a> <b>)</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Calibration factor can be read after calibration, using function HAL_ADC_GetValue() (value on 7 bits: from DR[6;0]).</li> </ul>

## 5.2 ADCEx Firmware driver defines

### 5.2.1 ADCEx

ADCEx

**ADCEx Private Constants**

ADC\_DISABLE\_TIMEOUT  
ADC\_CALIBRATION\_TIMEOUT

## 6 HAL CAN Generic Driver

### 6.1 CAN Firmware driver registers structures

#### 6.1.1 CAN\_InitTypeDef

*CAN\_InitTypeDef* is defined in the `stm32f0xx_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 CAN\_FilterConfTypeDef

*CAN\_FilterConfTypeDef* is defined in the `stm32f0xx_hal_can.h`



**Data Fields**

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

**Field Documentation**

- ***uint32\_t CAN\_FilterTypeDef::FilterIdHigh*** Specifies the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- ***uint32\_t CAN\_FilterTypeDef::FilterIdLow*** Specifies the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- ***uint32\_t CAN\_FilterTypeDef::FilterMaskIdHigh*** Specifies the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- ***uint32\_t CAN\_FilterTypeDef::FilterMaskIdLow*** Specifies the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- ***uint32\_t CAN\_FilterTypeDef::FilterFIFOAssignment*** Specifies the FIFO (0 or 1) which will be assigned to the filter. This parameter can be a value of **CAN\_filter\_FIFO**
- ***uint32\_t CAN\_FilterTypeDef::FilterNumber*** Specifies the filter which will be initialized. This parameter must be a number between Min\_Data = 0 and Max\_Data = 27.
- ***uint32\_t CAN\_FilterTypeDef::FilterMode*** Specifies the filter mode to be initialized. This parameter can be a value of **CAN\_filter\_mode**
- ***uint32\_t CAN\_FilterTypeDef::FilterScale*** Specifies the filter scale. This parameter can be a value of **CAN\_filter\_scale**
- ***uint32\_t CAN\_FilterTypeDef::FilterActivation*** Enable or disable the filter. This parameter can be set to ENABLE or DISABLE.
- ***uint32\_t CAN\_FilterTypeDef::BankNumber*** Select the start slave bank filter. This parameter must be a number between Min\_Data = 0 and Max\_Data = 28.

### 6.1.3 CanTxMsgTypeDef

**CanTxMsgTypeDef** is defined in the `stm32f0xx_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.

### 6.1.4 CanRxMsgTypeDef

*CanRxMsgTypeDef* is defined in the `stm32f0xx_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 CAN\_FIFO0 or CAN\_FIFO1

### 6.1.5 CAN\_HandleTypeDef

**CAN\_HandleTypeDef** is defined in the `stm32f0xx_hal_can.h`

#### Data Fields

- ***CAN\_TypeDef \* Instance***
- ***CAN\_InitTypeDef Init***
- ***CanTxMsgTypeDef \* pTxMsg***
- ***CanRxMsgTypeDef \* pRxMsg***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_CAN\_StateTypeDef State***
- ***\_\_IO HAL\_CAN\_ErrorTypeDef 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 HAL\_CAN\_ErrorTypeDef 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 `__CAN_CLK_ENABLE()`;
2. CAN pins configuration
  - Enable the clock for the CAN GPIOs using the following function:  
`__GPIOx_CLK_ENABLE();`
  - Connect and configure the involved CAN pins to AF9 using the following function  
`HAL_GPIO_Init();`
3. Initialise and configure the CAN using `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 ( <a href="#">CAN_HandleTypeDef</a> * 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 6.2.6 HAL\_CAN\_ConfigFilter

Function Name	<b>HAL_StatusTypeDef HAL_CAN_ConfigFilter ( <a href="#">CAN_HandleTypeDef</a> * hcan, <a href="#">CAN_FilterConfTypeDef</a> * 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 that contains the filter configuration information.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 6.2.7 HAL\_CAN\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_CAN_DeInit ( CAN_HandleTypeDef * hcan)</b>
Function Description	Deinitializes the CANx peripheral registers to their default reset values.
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><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 6.2.8 HAL\_CAN\_MspInit

Function Name	<b>void HAL_CAN_MspInit ( CAN_HandleTypeDef * hcan)</b>
Function Description	Initializes the CAN MSP.
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>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 6.2.9 HAL\_CAN\_MspDeInit

Function Name	<b>void HAL_CAN_MspDelInit ( <i>CAN_HandleTypeDef</i> * hcan)</b>
Function Description	Deinitializes the CAN MSP.
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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.10 HAL\_CAN\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Transmit ( <i>CAN_HandleTypeDef</i> * hcan, uint32_t Timeout)</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><li>• <b>Timeout</b> : Timeout duration.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.11 HAL\_CAN\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Transmit_IT ( <i>CAN_HandleTypeDef</i> * 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 6.2.12 HAL\_CAN\_Receive

Function Name	<code>HAL_StatusTypeDef HAL_CAN_Receive (</code> <code>CAN_HandleTypeDef * hcan, uint8_t FIFONumber, uint32_t</code> <code>Timeout)</code>
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.</li> <li>• <b>Timeout</b> : Timeout duration.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 6.2.13 HAL\_CAN\_Receive\_IT

Function Name	<code>HAL_StatusTypeDef HAL_CAN_Receive_IT (</code> <code>CAN_HandleTypeDef * hcan, uint8_t FIFONumber)</code>
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.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 6.2.14 HAL\_CAN\_Sleep

Function Name	<code>HAL_StatusTypeDef HAL_CAN_Sleep ( CAN_HandleTypeDef * hcan)</code>
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>• <b>HAL status.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.15 HAL\_CAN\_WakeUp

Function Name	<b>HAL_StatusTypeDef HAL_CAN_WakeUp (</b> <b>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>• <b>HAL status.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.16 HAL\_CAN\_IRQHandler

Function Name	<b>void HAL_CAN_IRQHandler (</b> <b>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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.17 HAL\_CAN\_TxCpltCallback

Function Name	<b>void HAL_CAN_TxCpltCallback (</b> <b>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>
Notes	<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>
Notes	<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>
Notes	<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>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 6.2.21 HAL\_CAN\_GetError

Function Name	<code>uint32_t HAL_CAN_GetError ( CAN_HandleTypeDef * hcan)</code>
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>• <b>CAN Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 6.3 CAN Firmware driver defines

### 6.3.1 CAN

CAN

*CAN clock prescaler*

`IS_CAN_PRESCALER`

*CAN Exported Macros*

`_HAL_CAN_RESET_HANDLE_STATE` **Description:**

- Reset CAN handle state.

**Parameters:**

- `_HANDLE_`: CAN handle.

**Return value:**

- None:

`_HAL_CAN_ENABLE_IT`

**Description:**

- Enable the specified CAN interrupts.

**Parameters:**

- `_HANDLE_`: CAN handle.

- `__INTERRUPT__`: CAN Interrupt

**Return value:**

- None:

`__HAL_CAN_DISABLE_IT`

**Description:**

- Disable the specified CAN interrupts.

**Parameters:**

- `__HANDLE__`: CAN handle.
- `__INTERRUPT__`: CAN 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.

`CAN_FLAG_MASK`

**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).

`__HAL_CAN_GET_FLAG`  
`__HAL_CAN_CLEAR_FLAG`

**Description:**

- Clear the specified CAN pending flag.

**Parameters:**

- `__HANDLE__`: specifies the CAN Handle.
- `__FLAG__`: specifies the flag to check.  
This parameter can be one of the following values:
  - CAN\_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\_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).

[\\_\\_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: FIFO0 message pending interrupt enable
  - CAN\_IT\_FMP1: FIFO1 message pending interrupt enable

**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\_\_: CAN handle.
- \_\_TRANSMITMAILBOX\_\_: the number of the mailbox that is used for transmission.

**Return value:**

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

[\\_\\_HAL\\_CAN\\_FIFO\\_RELEASE](#)**Description:**

- Release the specified receive FIFO.

**Parameters:**

- HANDLE: CAN handle.
- FIFONUMBER: Receive FIFO number, CAN\_FIFO0 or CAN\_FIFO1.

**Return value:**

- None:

\_HAL\_CAN\_CANCEL\_TRANSMIT

- Cancel a transmit request.

**Parameters:**

- HANDLE: specifies the CAN Handle.
- TRANSMITMAILBOX: the number of the mailbox that is used for transmission.

**Return value:**

- None:

\_HAL\_CAN\_DBG\_FREEZE

- 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:

**CAN filter FIFO**

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

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

IS\_CAN\_FILTER\_FIFO

CAN\_FilterFIFO0

CAN\_FilterFIFO1

**CAN filter mode**

CAN\_FILTERMODE\_IDMASK Identifier mask mode

CAN\_FILTERMODE\_IDLIST Identifier list mode

IS\_CAN\_FILTER\_MODE

**CAN filter number**

IS\_CAN\_FILTER\_NUMBER

**CAN filter scale**

CAN\_FILTERSCALE\_16BIT Two 16-bit filters

CAN\_FILTERSCALE\_32BIT One 32-bit filter

IS\_CAN\_FILTER\_SCALE

**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_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

**CAN identifier type**

CAN\_ID\_STD Standard Id

CAN\_ID\_EXT Extended Id

IS\_CAN\_IDTYPE

**CAN InitStatus**

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_IT_RQCP0	
CAN_IT_RQCP1	
CAN_IT_RQCP2	

***CAN Mailboxes***

CAN_TXMAILBOX_0
CAN_TXMAILBOX_1
CAN_TXMAILBOX_2

***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

**IS\_CAN\_MODE*****CAN Private Constants*****HAL\_CAN\_DEFAULT\_TIMEOUT*****CAN receive FIFO number constants***

CAN_FIFO0	CAN FIFO 0 used to receive
CAN_FIFO1	CAN FIFO 1 used to receive
<b>IS_CAN_FIFO</b>	

***CAN remote transmission request***

CAN_RTR_DATA	Data frame
CAN_RTR_REMOTE	Remote frame
<b>IS_CAN_RTR</b>	

***CAN Start bank filter for slave CAN*****IS\_CAN\_BANKNUMBER*****CAN synchronisation 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

IS\_CAN\_SJW

***CAN Timeouts***

INAK\_TIMEOUT

SLAK\_TIMEOUT

***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

IS\_CAN\_BS1

***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

IS\_CAN\_BS2

***CAN transmit constants***



---

CAN_TXSTATUS_FAILED	CAN transmission failed
CAN_TXSTATUS_OK	CAN transmission succeeded
CAN_TXSTATUS_PENDING	CAN transmission pending
CAN_TXSTATUS_NOMAILBOX	CAN cell did not provide CAN_TxStatus_NoMailBox

**CAN Tx**

IS\_CAN\_TRANSMITMAILBOX  
IS\_CAN\_STDID  
IS\_CAN\_EXTID  
IS\_CAN\_DLC

## 7 HAL CEC Generic Driver

### 7.1 CEC Firmware driver registers structures

#### 7.1.1 CEC\_InitTypeDef

**CEC\_InitTypeDef** is defined in the stm32f0xx\_hal\_cec.h

##### Data Fields

- *uint32\_t SignalFreeTime*
- *uint32\_t Tolerance*
- *uint32\_t BRERxStop*
- *uint32\_t BREErrorBitGen*
- *uint32\_t LBPEErrorBitGen*
- *uint32\_t BroadcastMsgNoErrorBitGen*
- *uint32\_t SignalFreeTimeOption*
- *uint32\_t OwnAddress*
- *uint32\_t ListenMode*
- *uint8\_t InitiatorAddress*

##### Field Documentation

- ***uint32\_t CEC\_InitTypeDef::SignalFreeTime*** Set SFT field, specifies the Signal Free Time. It can be one of **CEC\_Signal\_Free\_Time** and belongs to the set {0,...,7} where 0x0 is the default configuration else means 0.5 + (SignalFreeTime - 1) nominal data bit periods
- ***uint32\_t CEC\_InitTypeDef::Tolerance*** Set RXTOL bit, specifies the tolerance accepted on the received waveforms, it can be a value of **CEC\_Tolerance** : it is either CEC\_STANDARD\_TOLERANCE or CEC\_EXTENDED\_TOLERANCE
- ***uint32\_t CEC\_InitTypeDef::BRERxStop*** Set BRESTOP bit **CEC\_BRERxStop** : specifies whether or not a Bit Rising Error stops the reception.  
CEC\_NO\_RX\_STOP\_ON\_BRE: reception is not stopped. CEC\_RX\_STOP\_ON\_BRE: reception is stopped.
- ***uint32\_t CEC\_InitTypeDef::BREErrorBitGen*** Set BREGEN bit  
**CEC\_BREErrorBitGen** : specifies whether or not an Error-Bit is generated on the CEC line upon Bit Rising Error detection.  
CEC\_BRE\_ERRORBIT\_NO\_GENERATION: no error-bit generation.  
CEC\_BRE\_ERRORBIT\_GENERATION: error-bit generation if BRESTOP is set.
- ***uint32\_t CEC\_InitTypeDef::LBPEErrorBitGen*** Set LBPEGEN bit  
**CEC\_LBPEErrorBitGen** : specifies whether or not an Error-Bit is generated on the CEC line upon Long Bit Period Error detection.  
CEC\_LBPE\_ERRORBIT\_NO\_GENERATION: no error-bit generation.  
CEC\_LBPE\_ERRORBIT\_GENERATION: error-bit generation.
- ***uint32\_t CEC\_InitTypeDef::BroadcastMsgNoErrorBitGen*** Set BRDNOGEN bit  
**CEC\_BroadCastMsgErrorBitGen** : allows to avoid an Error-Bit generation on the CEC line upon an error detected on a broadcast message. It supersedes BREGEN and LBPEGEN bits for a broadcast message error handling. It can take two values:1) CEC\_BROADCASTERROR\_ERRORBIT\_GENERATION. a) BRE detection: error-bit generation on the CEC line if BRESTOP=CEC\_RX\_STOP\_ON\_BRE and BREGEN=CEC\_BRE\_ERRORBIT\_NO\_GENERATION. b) LBPE detection: error-bit generation on the CEC line if

- LBPGEN=CEC\_LBPE\_ERRORBIT\_NO\_GENERATION.2)  
 CEC\_BROADCASTERROR\_NO\_ERRORBIT\_GENERATION. no error-bit generation in case neither a) nor b) are satisfied. Additionally, there is no error-bit generation in case of Short Bit Period Error detection in a broadcast message while LSTN bit is set.
- ***uint32\_t CEC\_InitTypeDef::SignalFreeTimeOption*** Set SFTOP bit  
***CEC\_SFT\_Option*** : specifies when SFT timer starts. CEC\_SFT\_START\_ON\_TXSOM SFT: timer starts when TXSOM is set by software.  
***CEC\_SFT\_START\_ON\_TX\_RX\_END***: SFT timer starts automatically at the end of message transmission/reception.
  - ***uint32\_t CEC\_InitTypeDef::OwnAddress*** Set OAR field, specifies CEC device address within a 15-bit long field
  - ***uint32\_t CEC\_InitTypeDef::ListenMode*** Set LSTN bit ***CEC\_Listening\_Mode*** : specifies device listening mode. It can take two values:  
***CEC\_REDUCED\_LISTENING\_MODE***: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received.  
***CEC\_FULL\_LISTENING\_MODE***: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.
  - ***uint8\_t CEC\_InitTypeDef::InitiatorAddress***

### 7.1.2 CEC\_HandleTypeDef

***CEC\_HandleTypeDef*** is defined in the `stm32f0xx_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\_InitTypeDef CEC\_HandleTypeDef::Init***
- ***uint8\_t\* CEC\_HandleTypeDef::pTxBuffPtr***
- ***uint16\_t CEC\_HandleTypeDef::TxXferCount***
- ***uint8\_t\* CEC\_HandleTypeDef::pRxBuffPtr***
- ***uint16\_t CEC\_HandleTypeDef::RxXferSize***
- ***uint32\_t CEC\_HandleTypeDef::ErrorCode***
- ***HAL\_LockTypeDef CEC\_HandleTypeDef::Lock***
- ***HAL\_CEC\_StateTypeDef CEC\_HandleTypeDef::State***

## 7.2 CEC Firmware driver API description

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

## 7.2.1 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:
  - Enable the CEC interface clock.
  - CEC pins configuration:
    - Enable the clock for the CEC GPIOs.
    - Configure these CEC pins as alternate function pull-up.
  - NVIC configuration if you need to use interrupt process (HAL\_CEC\_Transmit\_IT() and HAL\_CEC\_Receive\_IT() APIs):
    - Configure the CEC interrupt priority.
    - Enable the NVIC CEC IRQ handle.
3. Program the Signal Free Time (SFT) and SFT option, Tolerance, reception stop in in case of Bit Rising Error, Error-Bit generation conditions, device logical address and Listen mode in the hcec Init structure.
4. Initialize the CEC registers by calling the HAL\_CEC\_Init() API.
  - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc by calling the customized HAL\_CEC\_MspInit() API. The specific CEC interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros \_\_HAL\_CEC\_ENABLE\_IT() and \_\_HAL\_CEC\_DISABLE\_IT() inside the transmit and receive process.

## 7.2.2 Initialization and Configuration functions

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

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

## 7.2.3 IO operation function

- [\*\*HAL\\_CEC\\_Transmit\(\)\*\*](#)
- [\*\*HAL\\_CEC\\_Receive\(\)\*\*](#)
- [\*\*HAL\\_CEC\\_Transmit\\_IT\(\)\*\*](#)
- [\*\*HAL\\_CEC\\_Receive\\_IT\(\)\*\*](#)
- [\*\*HAL\\_CEC\\_IRQHandler\(\)\*\*](#)

- [\*\*\*HAL\\_CEC\\_TxCpltCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_CEC\\_RxCpltCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_CEC\\_ErrorCallback\(\)\*\*\*](#)

#### 7.2.4 Peripheral Control function

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

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

#### 7.2.5 HAL\_CEC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Init ( <a href="#"><b><i>CEC_HandleTypeDef</i></b></a> * hcec)</b>
Function Description	Initializes the CEC mode according to the specified parameters in the <b>CEC_InitTypeDef</b> 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 7.2.6 HAL\_CEC\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_CEC_DelInit ( <a href="#"><b><i>CEC_HandleTypeDef</i></b></a> * 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 7.2.7 HAL\_CEC\_MsplInit

---

Function Name	<b>void HAL_CEC_MspInit ( <i>CEC_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 7.2.8 HAL\_CEC\_MspDelInit

Function Name	<b>void HAL_CEC_MspDelInit ( <i>CEC_HandleTypeDef</i> * hcec)</b>
Function Description	CEC MSP Delinit.
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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 7.2.9 HAL\_CEC\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Transmit ( <i>CEC_HandleTypeDef</i> * 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 7.2.10 HAL\_CEC\_Receive

Function Name	<code>HAL_StatusTypeDef HAL_CEC_Receive (</code> <code>    <b>CEC_HandleTypeDef</b> * hcec, uint8_t * pData, uint32_t</code> <code>    Timeout)</code>
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. Note that 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 = 0</code></li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 7.2.11 HAL\_CEC\_Transmit\_IT

Function Name	<code>HAL_StatusTypeDef HAL_CEC_Transmit_IT (</code> <code>    <b>CEC_HandleTypeDef</b> * hcec, uint8_t DestinationAddress,</code> <code>    uint8_t * pData, uint32_t Size)</code>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 7.2.12 HAL\_CEC\_Receive\_IT

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

### 7.2.13 HAL\_CEC\_IRQHandler

Function Name	<b>void HAL_CEC_IRQHandler ( <i>CEC_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 7.2.14 HAL\_CEC\_TxCpltCallback

Function Name	<b>void HAL_CEC_TxCpltCallback ( <i>CEC_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 7.2.15 HAL\_KEC\_RxCpltCallback

Function Name	<b>void HAL_KEC_RxCpltCallback ( <i>CEC_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 7.2.16 HAL\_KEC\_ErrorCallback

Function Name	<b>void HAL_KEC_ErrorCallback ( <i>CEC_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 7.2.17 HAL\_KEC\_GetState

Function Name	<b>HAL_KEC_StateTypeDef HAL_KEC_GetState ( <i>CEC_HandleTypeDef</i> * 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>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 7.2.18 HAL\_CEC\_GetError

Function Name	<code>uint32_t HAL_CEC_GetError ( CEC_HandleTypeDef * hcec)</code>
Function Description	Return the CEC error code.
Parameters	<ul style="list-style-type: none"> <li>• <code>hcec</code> : 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>• <b>CEC Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 7.3 CEC Firmware driver defines

### 7.3.1 CEC

CEC

*all RX or TX errors flags in CEC ISR register*

`CEC_ISR_ALL_ERROR`

*Error Bit Generation if Bit Rise Error reported*

`CEC_BRE_ERRORBIT_NO_GENERATION`

`CEC_BRE_ERRORBIT_GENERATION`

`IS_CEC_BREERRORBITGEN`

*Reception Stop on Error*

`CEC_NO_RX_STOP_ON_BRE`

`CEC_RX_STOP_ON_BRE`

`IS_CEC_BRERXSTOP`

*Error Bit Generation on Broadcast message*

`CEC_BROADCASTERROR_ERRORBIT_GENERATION`

`CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION`

`IS_CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION`

**CEC Exported Macros**

`_HAL_CEC_RESET_HANDLE_STATE`

**Description:**

- Reset CEC handle state.

**Parameters:**

- `_HANDLE_`: CEC handle.

**Return value:**

- None:

### \_HAL\_CEC\_GET\_IT

#### **Description:**

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

#### **Parameters:**

- HANDLE: specifies the CEC Handle.
- INTERRUPT: specifies the interrupt to check. This parameter can be one of the following values:
  - CEC\_ISR\_RXBR : Rx-Byte Received
  - CEC\_ISR\_RXEND : End of Reception
  - CEC\_ISR\_RXOVR : Rx Overrun
  - CEC\_ISR\_BRE : Rx Bit Rising Error
  - CEC\_ISR\_SBPE : Rx Short Bit Period Error
  - CEC\_ISR\_LBPE : Rx Long Bit Period Error
  - CEC\_ISR\_RXACKE : Rx Missing Acknowledge
  - CEC\_ISR\_ARBLST : Arbitration lost
  - CEC\_ISR\_TXBR : Tx-Byte Request
  - CEC\_ISR\_TXEND : End of Transmission
  - CEC\_ISR\_TXUDR : Tx-buffer Underrun
  - CEC\_ISR\_TXERR : Tx Error
  - CEC\_ISR\_TXACKE : Tx Missing Acknowledge

#### **Return value:**

- ITStatus:

### \_HAL\_CEC\_CLEAR\_FLAG

#### **Description:**

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

#### **Parameters:**

- HANDLE: specifies the CEC Handle.
- FLAG: specifies the interrupt/status flag to clear. This parameter can be one of

the following values:

- CEC\_ISR\_RXBR : Rx-Byte Received
- CEC\_ISR\_RXEND : End of Reception
- CEC\_ISR\_RXOVR : Rx Overrun
- CEC\_ISR\_BRE : Rx Bit Rising Error
- CEC\_ISR\_SBPE : Rx Short Bit Period Error
- CEC\_ISR\_LBPE : Rx Long Bit Period Error
- CEC\_ISR\_RXACKE : Rx Missing Acknowledge
- CEC\_ISR\_ARBLST : Arbitration lost
- CEC\_ISR\_TXBR : Tx-Byte Request
- CEC\_ISR\_TXEND : End of Transmission
- CEC\_ISR\_TXUDR : Tx-buffer Underrun
- CEC\_ISR\_TXERR : Tx Error
- CEC\_ISR\_TXACKE : Tx Missing Acknowledge

#### **Return value:**

- none:

### [\\_\\_HAL\\_CEC\\_ENABLE\\_IT](#)

#### **Description:**

- Enables the specified CEC interrupt.

#### **Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the CEC Handle.
- [\\_\\_INTERRUPT\\_\\_](#): specifies the CEC interrupt to enable. This parameter can be one of the following values:
  - CEC\_IER\_RXBRIE : Rx-Byte Received IT Enable
  - CEC\_IER\_RXENDIE : End Of Reception IT Enable
  - CEC\_IER\_RXOVRIE : Rx-Overrun IT Enable
  - CEC\_IER\_BREIE : Rx Bit Rising Error IT Enable
  - CEC\_IER\_SBPEIE : Rx Short Bit period Error IT Enable

- CEC\_IER\_LBPEIE : Rx Long Bit period Error IT Enable
- CEC\_IER\_RXACKEIE : Rx Missing Acknowledge IT Enable
- CEC\_IER\_ARBLSTIE : Arbitration Lost IT Enable
- CEC\_IER\_TXBRIE : Tx Byte Request IT Enable
- CEC\_IER\_TXENDIE : End of Transmission IT Enable
- CEC\_IER\_TXUDRIE : Tx-Buffer Underrun IT Enable
- CEC\_IER\_TXERRIE : Tx-Error IT Enable
- CEC\_IER\_TXACKEIE : Tx Missing Acknowledge IT Enable

**Return value:**

- none:

**\_HAL\_CEC\_DISABLE\_IT****Description:**

- Disables the specified CEC interrupt.

**Parameters:**

- HANDLE: specifies the CEC Handle.
- INTERRUPT: specifies the CEC interrupt to disable. This parameter can be one of the following values:
  - CEC\_IER\_RXBRIE : Rx-Byte Received IT Enable
  - CEC\_IER\_RXENDIE : End Of Reception IT Enable
  - CEC\_IER\_RXOVRIE : Rx-Overrun IT Enable
  - CEC\_IER\_BREIE : Rx Bit Rising Error IT Enable
  - CEC\_IER\_SBPEIE : Rx Short Bit period Error IT Enable
  - CEC\_IER\_LBPEIE : Rx Long Bit period Error IT Enable
  - CEC\_IER\_RXACKEIE : Rx Missing Acknowledge IT Enable
  - CEC\_IER\_ARBLSTIE : Arbitration Lost IT Enable

- CEC\_IER\_TXBRIE : Tx Byte Request IT Enable
- CEC\_IER\_TXENDIE : End of Transmission IT Enable
- CEC\_IER\_RXUDRIE : Rx-Buffer Underrun IT Enable
- CEC\_IER\_RXERRIE : Rx-Error IT Enable
- CEC\_IER\_RXACKEIE : Rx Missing Acknowledge IT Enable

**Return value:**

- none:

[\\_\\_HAL\\_CEC\\_GET\\_IT\\_SOURCE](#)

**Description:**

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

**Parameters:**

- [\\_\\_HANDLE](#): specifies the CEC Handle.
- [\\_\\_INTERRUPT](#): specifies the CEC interrupt to check. This parameter can be one of the following values:
  - CEC\_IER\_RXBRIE : Rx-Byte Received IT Enable
  - CEC\_IER\_RXENDIE : End Of Reception IT Enable
  - CEC\_IER\_RXOVRIE : Rx-Overrun IT Enable
  - CEC\_IER\_BREIE : Rx Bit Rising Error IT Enable
  - CEC\_IER\_SBPEIE : Rx Short Bit period Error IT Enable
  - CEC\_IER\_LBPEIE : Rx Long Bit period Error IT Enable
  - CEC\_IER\_RXACKEIE : Rx Missing Acknowledge IT Enable
  - CEC\_IER\_ARBLSTIE : Arbitration Lost IT Enable
  - CEC\_IER\_TXBRIE : Tx Byte Request IT Enable
  - CEC\_IER\_TXENDIE : End of Transmission IT Enable
  - CEC\_IER\_RXUDRIE : Rx-Buffer Underrun IT Enable
  - CEC\_IER\_RXERRIE : Rx-Error IT Enable

- Error IT Enable
  - CEC\_IER\_TXACKIE : Tx Missing Acknowledge IT 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:**

- Set Transmission End flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the CEC Handle.

**Return value:**

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

**Description:**

- Get Transmission Start flag.

\_\_HAL\_CEC\_ENABLE

\_\_HAL\_CEC\_DISABLE

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

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

G  
\_\_HAL\_CEC\_GET\_TRANSMISSION\_START\_FLA

**Parameters:**

- \_\_HANDLE\_\_: specifies the CEC Handle.

**Return value:**

- FlagStatus:

**Description:**

- Get Transmission End flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the CEC Handle.

**Return value:**

- FlagStatus:

**Description:**

- Clear OAR register.

**Parameters:**

- \_\_HANDLE\_\_: specifies the CEC Handle.

**Return value:**

- none:

**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the CEC Handle.
- \_\_ADDRESS\_\_: Own Address value (CEC logical address is identified by bit position)

**Return value:**

- none:

**Description:**

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

**Parameters:**

- \_\_ADDRESS\_\_: CEC own address.

**Return value:**

IS\_KEC\_ADDRESS

- Test: result (TRUE or FALSE).

**Description:**

- Check CEC initiator or destination logical address setting.

**Parameters:**

- ADDRESS: CEC initiator or logical address.

**Return value:**

- Test: result (TRUE or FALSE).

IS\_KEC\_MSGSIZE

**Description:**

- Check CEC message size.

**Parameters:**

- SIZE: CEC message size.

**Return value:**

- Test: result (TRUE or FALSE).

***all RX errors interrupts enabling flag***

CEC\_IER\_RX\_ALL\_ERR

***all TX errors interrupts enabling flag***

CEC\_IER\_TX\_ALL\_ERR

***Initiator logical address position in message header***

CEC\_INITIATOR\_LSB\_POS

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

CEC\_LBPE\_ERRORBIT\_NO\_GENERATION

CEC\_LBPE\_ERRORBIT\_GENERATION

IS\_KEC\_LBPEERRORBITGEN

***Listening mode option***

CEC\_REDUCED\_LISTENING\_MODE

CEC\_FULL\_LISTENING\_MODE

IS\_KEC\_LISTENING\_MODE

***Device Own Address position in CEC CFGR register***

CEC\_CFGR\_OAR\_LSB\_POS

***CEC Private Constants***

CEC\_CFGR\_FIELDS

***Signal Free Time start option***

CEC\_SFT\_START\_ON\_TXSOM

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

IS\_CEC\_SFTOP

***Signal Free Time setting parameter***

CEC\_DEFAULT\_SFT

CEC\_0\_5\_BITPERIOD\_SFT

CEC\_1\_5\_BITPERIOD\_SFT

CEC\_2\_5\_BITPERIOD\_SFT

CEC\_3\_5\_BITPERIOD\_SFT

CEC\_4\_5\_BITPERIOD\_SFT

CEC\_5\_5\_BITPERIOD\_SFT

CEC\_6\_5\_BITPERIOD\_SFT

IS\_CEC\_SIGNALFREETIME

***Receiver Tolerance***

CEC\_STANDARD\_TOLERANCE

CEC\_EXTENDED\_TOLERANCE

IS\_CEC\_TOLERANCE

## 8 HAL COMP Generic Driver

### 8.1 COMP Firmware driver registers structures

#### 8.1.1 COMP\_InitTypeDef

**COMP\_InitTypeDef** is defined in the `stm32f0xx_hal_comp.h`

##### Data Fields

- `uint32_t InvertingInput`
- `uint32_t NonInvertingInput`
- `uint32_t Output`
- `uint32_t OutputPol`
- `uint32_t Hysteresis`
- `uint32_t Mode`
- `uint32_t WindowMode`
- `uint32_t TriggerMode`

##### Field Documentation

- `uint32_t COMP_InitTypeDef::InvertingInput` Selects the inverting input of the comparator. This parameter can be a value of **COMP\_InvertingInput**
- `uint32_t COMP_InitTypeDef::NonInvertingInput` Selects the non inverting input of the comparator. This parameter can be a value of **COMP\_NonInvertingInput**
- `uint32_t COMP_InitTypeDef::Output` Selects the output redirection of the comparator. This parameter can be a value of **COMP\_Output**
- `uint32_t COMP_InitTypeDef::OutputPol` Selects the output polarity of the comparator. This parameter can be a value of **COMP\_OutputPolarity**
- `uint32_t COMP_InitTypeDef::Hysteresis` Selects the hysteresis voltage of the comparator. This parameter can be a value of **COMP\_Hysteresis**
- `uint32_t COMP_InitTypeDef::Mode` Selects the operating consumption mode of the comparator to adjust the speed/consumption. This parameter can be a value of **COMP\_Mode**
- `uint32_t COMP_InitTypeDef::WindowMode` Selects the window mode of the comparator 1 & 2. This parameter can be a value of **COMP\_WindowMode**
- `uint32_t COMP_InitTypeDef::TriggerMode` Selects the trigger mode of the comparator (interrupt mode). This parameter can be a value of **COMP\_TriggerMode**

#### 8.1.2 COMP\_HandleTypeDef

**COMP\_HandleTypeDef** is defined in the `stm32f0xx_hal_comp.h`

##### Data Fields

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

##### Field Documentation

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

## 8.2 COMP Firmware driver API description

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

### 8.2.1 COMP Peripheral features

The STM32F0xx device family integrates up to 2 analog comparators COMP1 and COMP2:

1. The non inverting input and inverting input can be set to GPIO pins as shown in table1. COMP Inputs below.
2. The COMP output is available using `HAL_COMP_GetOutputLevel()` and can be set on GPIO pins. Refer to table 2. COMP Outputs below.
3. The COMP output can be redirected to embedded timers (TIM1, TIM2 and TIM3) Refer to table 3. COMP Outputs redirection to embedded timers below.
4. The comparators COMP1 and COMP2 can be combined in window mode.
5. The comparators have interrupt capability with wake-up from Sleep and Stop modes (through the EXTI controller):
  - COMP1 is internally connected to EXTI Line 21
  - COMP2 is internally connected to EXTI Line 22 From the corresponding IRQ handler, the right interrupt source can be retrieved with the macro `__HAL_COMP_EXTI_GET_FLAG()`. Possible values are:
    - `COMP_EXTI_LINE_COMP1_EVENT`
    - `COMP_EXTI_LINE_COMP2_EVENT`

Table 16: COMP Inputs for STM32F05xx, STM32F07x and STM32F09x devices

		COMP1	COMP2
Inverting inputs	1/4 VREFINT	OK	OK
	1/2 VREFINT	OK	OK
	3/4 VREFINT	OK	OK
	VREFINT	OK	OK
	DAC1 OUT (PA4)	OK	OK
	DAC2 OUT (PA5)	OK	OK
	I/O1	PA0	PA2
Non-inverting inputs		PA1	PA3

Table 17: COMP outputs for STM32F05xx, STM32F07x and STM32F09x devices

COMP1	COMP2
PA0	PA2
PA6	PA7
PA11	PA12

**Table 18: Redirection of COMP outputs to embedded timers for STM32F05xx, STM32F07x and STM32F09x devices**

COMP1	COMP2
TIM1 BKIN	TIM1 BKIN
TIM1 OCREFCLR	TIM1 OCREFCLR
TIM1 IC1	TIM1 IC1
TIM2 IC4	TIM2 IC4
TIM2 OCREFCLR	TIM2 OCREFCLR
TIM3 IC1	TIM3 IC3
TIM3 OCREFCLR	TIM3 OCREFCLR

## 8.2.2 How to use this driver

This driver provides functions to configure and program the Comparators of STM32F05x, STM32F07x and STM32F09x devices. To use the comparator, perform the following steps:

1. Fill in the HAL\_COMP\_MspInit() to
  - Configure the comparator input in analog mode using HAL\_GPIO\_Init()
  - Configure the comparator output in alternate function mode using HAL\_GPIO\_Init() to map the comparator output to the GPIO pin
  - If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using HAL\_GPIO\_Init() function. After that enable the comparator interrupt vector using HAL\_NVIC\_EnableIRQ() function.
2. Configure the comparator using HAL\_COMP\_Init() function:
  - Select the inverting input
  - Select the non inverting input
  - Select the output polarity
  - Select the output redirection
  - Select the hysteresis level
  - Select the power mode
  - Select the event/interrupt mode
3. Enable the comparator using HAL\_COMP\_Start() function or HAL\_COMP\_Start\_IT() function for interrupt mode
4. Read the comparator output level with HAL\_COMP\_GetOutputLevel()

## 8.2.3 Initialization and Configuration functions

This section provides functions to initialize and de-initialize comparators

- [\*\*HAL\\_COMP\\_Init\(\)\*\*](#)
- [\*\*HAL\\_COMP\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_COMP\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_COMP\\_MspDelInit\(\)\*\*](#)

## 8.2.4 IO operation functions

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

- `HAL_COMP_Start()`
- `HAL_COMP_Stop()`
- `HAL_COMP_Start_IT()`
- `HAL_COMP_Stop_IT()`
- `HAL_COMP_IRQHandler()`

## 8.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the COMP data transfers.

- `HAL_COMP_Lock()`
- `HAL_COMP_GetOutputLevel()`
- `HAL_COMP_TriggerCallback()`

## 8.2.6 Peripheral State functions

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

- `HAL_COMP_GetState()`

## 8.2.7 HAL\_COMP\_Init

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

## 8.2.8 HAL\_COMP\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_COMP_DeInit ( COMP_HandleTypeDef * hcomp )</code>
Function Description	Deinitializes the COMP peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcomp</b> : COMP handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

**Notes**

- Deinitialization can't be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

### 8.2.9 HAL\_COMP\_MspInit

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

### 8.2.10 HAL\_COMP\_MspDeInit

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

### 8.2.11 HAL\_COMP\_Start

Function Name	<b>HAL_StatusTypeDef HAL_COMP_Start ( <i>COMP_HandleTypeDef</i> * hcomp)</b>
Function Description	Start the comparator.
Parameters	<ul style="list-style-type: none"><li>• <b>hcomp</b> : COMP handle</li></ul>

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

### 8.2.12 HAL\_COMP\_Stop

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

### 8.2.13 HAL\_COMP\_Start\_IT

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

### 8.2.14 HAL\_COMP\_Stop\_IT

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

---

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

### 8.2.15 HAL\_COMP\_IRQHandler

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

### 8.2.16 HAL\_COMP\_Lock

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

### 8.2.17 HAL\_COMP\_GetOutputLevel

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

## 8.2.18 HAL\_COMP\_TriggerCallback

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

## 8.2.19 HAL\_COMP\_GetState

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

## 8.3 COMP Firmware driver defines

### 8.3.1 COMP

COMP

**COMP Exported Macros**

<code>_HAL_COMP_RESET_HANDLE_STATE</code>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Reset COMP handle state.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: COMP handle.</li> </ul>

**Return value:**

- None:

**Description:**

- Checks whether the specified EXTI line flag is set or not.

**Parameters:**

- FLAG: specifies the COMP Exti sources to be checked. This parameter can be a value of

**Return value:**

- The state of FLAG (SET or RESET).

\_HAL\_COMP\_EXTI\_GET\_FLAG

**Description:**

- Clear the COMP Exti flags.

**Parameters:**

- FLAG: specifies the COMP Exti sources to be cleared. This parameter can be a value of

**Return value:**

- None.:

\_HAL\_COMP\_EXTI\_ENABLE\_IT

**Description:**

- Enable the COMP Exti Line.

**Parameters:**

- EXTILINE: specifies the COMP Exti sources to be enabled. This parameter can be a value of

**Return value:**

- None.:

\_HAL\_COMP\_EXTI\_DISABLE\_IT

**Description:**

- Disable the COMP Exti Line.

**Parameters:**

- EXTILINE: specifies the COMP Exti sources to be disabled. This parameter can be a value of

**Return value:**

- None.:

\_HAL\_COMP\_EXTI\_RISING\_IT\_ENABLE

**Description:**

- Enable the Exti Line rising edge trigger.

**Parameters:**

- `__EXTILINE__`: specifies the COMP Exti sources to be enabled. This parameter can be a value of

**Return value:**

- None.:

`__HAL_COMP_EXTI_RISING_IT_DISABLE`**Description:**

- Disable the Exti Line rising edge trigger.

**Parameters:**

- `__EXTILINE__`: specifies the COMP Exti sources to be disabled. This parameter can be a value of

**Return value:**

- None.:

`__HAL_COMP_EXTI_FALLING_IT_ENABLE`**Description:**

- Enable the Exti Line falling edge trigger.

**Parameters:**

- `__EXTILINE__`: specifies the COMP Exti sources to be enabled. This parameter can be a value of

**Return value:**

- None.:

`__HAL_COMP_EXTI_FALLING_IT_DISABLE`**Description:**

- Disable the Exti Line falling edge trigger.

**Parameters:**

- `__EXTILINE__`: specifies the COMP Exti sources to be disabled. This parameter can be a value of

**Return value:**

- None.:

`__HAL_COMP_GET_EXTI_LINE`**Description:**

- Get the specified EXTI line for a comparator instance.

**Parameters:**

- `__INSTANCE__`: specifies the COMP instance.

**Return value:**

- value: of

***COMP ExtiLineEvent***

COMP_EXTI_LINE_COMP1_EVENT	External interrupt line 21 Connected to COMP1
COMP_EXTI_LINE_COMP2_EVENT	External interrupt line 22 Connected to COMP2
<b><i>COMP Hysteresis</i></b>	
COMP_HYSTERESIS_NONE	No hysteresis
COMP_HYSTERESIS_LOW	Hysteresis level low
COMP_HYSTERESIS_MEDIUM	Hysteresis level medium
COMP_HYSTERESIS_HIGH	Hysteresis level high
IS_COMP_HYSTERESIS	
<b><i>COMP InvertingInput</i></b>	
COMP_INVERTINGINPUT_1_4VREFINT	1/4 VREFINT connected to comparator inverting input
COMP_INVERTINGINPUT_1_2VREFINT	1/2 VREFINT connected to comparator inverting input
COMP_INVERTINGINPUT_3_4VREFINT	3/4 VREFINT connected to comparator inverting input
COMP_INVERTINGINPUT_VREFINT	VREFINT connected to comparator inverting input
COMP_INVERTINGINPUT_DAC1	DAC_OUT1 (PA4) connected to comparator inverting input
COMP_INVERTINGINPUT_DAC1SWITCHCLOSED	DAC_OUT1 (PA4) connected to comparator inverting input and close switch (PA0 for COMP1 only)
COMP_INVERTINGINPUT_DAC2	DAC_OUT2 (PA5) connected to comparator inverting input
COMP_INVERTINGINPUT_IO1	IO (PA0 for COMP1 and PA2 for COMP2) connected to comparator inverting input
IS_COMP_INVERTINGINPUT	
<b><i>COMP Lock</i></b>	
COMP_LOCK_DISABLE	
COMP_LOCK_ENABLE	
COMP_STATE_BIT_LOCK	
<b><i>COMP Mode</i></b>	
COMP_MODE_HIGHSPEED	High Speed
COMP_MODE_MEDIUMSPEED	Medium Speed
COMP_MODE_LOWPOWER	Low power mode
COMP_MODE_ULTRALOWPOWER	Ultra-low power mode
IS_COMP_MODE	
<b><i>COMP NonInvertingInput</i></b>	
COMP_NONINVERTINGINPUT_IO1	I/O1 (PA1 for COMP1, PA3 for

	COMP2) connected to comparator non inverting input
COMP_NONINVERTINGINPUT_DAC1SWITCHCLOSED	DAC ouput connected to comparator COMP1 non inverting input
<b>IS_COMP_NONINVERTINGINPUT</b>	
<b>COMP Output</b>	
COMP_OUTPUT_NONE	COMP output isn't connected to other peripherals
COMP_OUTPUT_TIM1BKin	COMP output connected to TIM1 Break Input (BKIN)
COMP_OUTPUT_TIM1IC1	COMP output connected to TIM1 Input Capture 1
COMP_OUTPUT_TIM1OCREFCLR	COMP output connected to TIM1 OCREF Clear
COMP_OUTPUT_TIM2IC4	COMP output connected to TIM2 Input Capture 4
COMP_OUTPUT_TIM2OCREFCLR	COMP output connected to TIM2 OCREF Clear
COMP_OUTPUT_TIM3IC1	COMP output connected to TIM3 Input Capture 1
COMP_OUTPUT_TIM3OCREFCLR	COMP output connected to TIM3 OCREF Clear
<b>IS_COMP_OUTPUT</b>	
<b>COMP OutputLevel</b>	
COMP_OUTPUTLEVEL_LOW	
COMP_OUTPUTLEVEL_HIGH	
<b>COMP OutputPolarity</b>	
COMP_OUTPUTPOL_NONINVERTED	COMP output on GPIO isn't inverted
COMP_OUTPUTPOL_INVERTED	COMP output on GPIO is inverted
<b>IS_COMP_OUTPUTPOL</b>	
<b>COMP Private Constants</b>	
COMP_CSR_RESET_VALUE	
COMP_CSR_RESET_PARAMETERS_MASK	
COMP_CSR_UPDATE_PARAMETERS_MASK	
COMP_CSR_COMPxNONINSEL_MASK	
COMP_CSR_COMP1_SHIFT	
COMP_CSR_COMP2_SHIFT	
<b>COMP TriggerMode</b>	
COMP_TRIGGERMODE_NONE	No External Interrupt trigger detection
COMP_TRIGGERMODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
COMP_TRIGGERMODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
COMP_TRIGGERMODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection

IS\_COMP\_TRIGGERMODE

**COMP WindowMode**

COMP\_WINDOWMODE\_DISABLED Window mode disabled

COMP\_WINDOWMODE\_ENABLED Window mode enabled: non inverting input of comparator 2 is connected to the non inverting input of comparator 1 (PA1)

IS\_COMP\_WINDOWMODE

## 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 How to use this driver

##### How to configure Interrupts using CORTEX HAL driver

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

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

##### How to configure Systick using CORTEX HAL driver

Setup SysTick Timer for time base

- The HAL\_SYSTICK\_Config() function calls the SysTick\_Config() function which is a CMSIS function that:
  - Configures the SysTick Reload register with value passed as function parameter.
  - Configures the SysTick IRQ priority to the lowest value (0x03).
  - Resets the SysTick Counter register.
  - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
  - Enables the SysTick Interrupt.
  - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK\_Div8 by calling the macro \_\_HAL\_CORTEX\_SYSTICKCLK\_CONFIG(SYSTICK\_CLKSOURCE\_HCLK\_DIV8) just after the HAL\_SYSTICK\_Config() function call. The \_\_HAL\_CORTEX\_SYSTICKCLK\_CONFIG() macro is defined inside the stm32f0xx\_hal\_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL\_NVIC\_SetPriority(SysTick\_IRQn,...) function just after the HAL\_SYSTICK\_Config() function call. The HAL\_NVIC\_SetPriority() call the NVIC\_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL\_SYSTICK\_Config() function
  - Reload Value should not exceed 0xFFFFFFF

#### 9.1.2 Initialization and de-initialization functions

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

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

### 9.1.3 Peripheral Control functions

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

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

### 9.1.4 HAL\_NVIC\_SetPriority

Function Name	<code>void HAL_NVIC_SetPriority ( IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)</code>
Function Description	Sets the priority of an interrupt.
Parameters	<ul style="list-style-type: none"> <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 stm32l0xx.h file)</li> <li>• <b>PreemptPriority</b> : The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 3. A lower priority value indicates a higher priority</li> <li>• <b>SubPriority</b> : The subpriority level for the IRQ channel. with stm32f0xx devices, this parameter is a dummy value and it is ignored, because no subpriority supported in Cortex M0 based products.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<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 (stm32f0xxxx.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 (stm32f0xxxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<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>
Notes	<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>• <b>status : - 0 Function succeeded.</b> <ul style="list-style-type: none"> <li>– <b>1 Function failed.</b></li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.1.9 HAL\_NVIC\_GetPriority

Function Name	<b>uint32_t HAL_NVIC_GetPriority ( IRQn_Type IRQn)</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 (stm32f0xxxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.1.10 HAL\_NVIC\_SetPendingIRQ

Function Name	<b>void HAL_NVIC_SetPendingIRQ ( IRQn_Type IRQn)</b>
Function Description	Sets Pending bit of an external 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 (stm32f0xxxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## Notes

- None.

**9.1.11 HAL\_NVIC\_GetPendingIRQ**

Function Name	<b>uint32_t HAL_NVIC_GetPendingIRQ ( IRQn_Type IRQn)</b>
Function Description	Gets Pending Interrupt (reads the pending register in the NVIC and returns the pending bit for the specified interrupt).
Parameters	<ul style="list-style-type: none"> <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 (stm32f0xxxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>status : - 0 Interrupt status is not pending.</b> <ul style="list-style-type: none"> <li>– <b>1 Interrupt status is pending.</b></li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**9.1.12 HAL\_NVIC\_ClearPendingIRQ**

Function Name	<b>void HAL_NVIC_ClearPendingIRQ ( IRQn_Type IRQn)</b>
Function Description	Clears the pending bit of an external 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 (stm32f0xxxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**9.1.13 HAL\_SYSTICK\_CLKSourceConfig**

Function Name	<b>void HAL_SYSTICK_CLKSourceConfig ( uint32_t CLKSource)</b>
---------------	---------------------------------------------------------------

Function Description	Configures the SysTick clock source.
Parameters	<ul style="list-style-type: none"><li>• <b>CLKSource</b> : specifies the SysTick clock source. This parameter can be one of the following values:<ul style="list-style-type: none"><li>– <b>SYSTICK_CLKSOURCE_HCLK_DIV8</b> AHB clock divided by 8 selected as SysTick clock source.</li><li>– <b>SYSTICK_CLKSOURCE_HCLK</b> AHB clock selected as SysTick clock source.</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 9.1.14 HAL\_SYSTICK\_IRQHandler

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

### 9.1.15 HAL\_SYSTICK\_Callback

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

## 9.2 CORTEX Firmware driver defines

### 9.2.1 CORTEX

CORTEX

***CORTEX Exported Macro***

`__HAL_CORTEX_SYSTICKCLK_CON  
FIG`

**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 Priority**

`IS_NVIC_PREEMPTION_PRIORITY`

**CORTEX SysTick clock source**

`SYSTICK_CLKSOURCE_HCLK_DIV8`

`SYSTICK_CLKSOURCE_HCLK`

`IS_SYSTICK_CLK_SOURCE`

## 10 HAL CRC Generic Driver

### 10.1 CRC Firmware driver registers structures

#### 10.1.1 CRC\_InitTypeDef

*CRC\_InitTypeDef* is defined in the `stm32f0xx_hal_crc.h`

##### Data Fields

- *uint8\_t DefaultPolynomialUse*
- *uint8\_t DefaultInitValueUse*
- *uint32\_t GeneratingPolynomial*
- *uint32\_t CRCLength*
- *uint32\_t InitValue*
- *uint32\_t InputDataInversionMode*
- *uint32\_t OutputDataInversionMode*

##### Field Documentation

- ***uint8\_t CRC\_InitTypeDef::DefaultPolynomialUse*** This parameter is a value of [\*\*\*CRC\\_Default\\_Polynomial\*\*\*](#) and indicates if default polynomial is used. If set to DEFAULT\_POLYNOMIAL\_ENABLE, resort to default  $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$ . In that case, there is no need to set GeneratingPolynomial field. If otherwise set to DEFAULT\_POLYNOMIAL\_DISABLE, GeneratingPolynomial and CRCLength fields must be set
- ***uint8\_t CRC\_InitTypeDef::DefaultInitValueUse*** This parameter is a value of [\*\*\*CRC\\_Default\\_InitValue\\_Use\*\*\*](#) and indicates if default init value is used. If set to DEFAULT\_INIT\_VALUE\_ENABLE, resort to default 0xFFFFFFFF value. In that case, there is no need to set InitValue field. If otherwise set to DEFAULT\_INIT\_VALUE\_DISABLE, InitValue field must be set
- ***uint32\_t CRC\_InitTypeDef::GeneratingPolynomial*** Set CRC generating polynomial. 7, 8, 16 or 32-bit long value for a polynomial degree respectively equal to 7, 8, 16 or 32. This field is written in normal representation, e.g., for a polynomial of degree 7,  $X^7 + X^6 + X^5 + X^2 + 1$  is written 0x65. No need to specify it if DefaultPolynomialUse is set to DEFAULT\_POLYNOMIAL\_ENABLE
- ***uint32\_t CRC\_InitTypeDef::CRCLength*** This parameter is a value of [\*\*\*CRCEx\\_Polynomial\\_Size\\_Definitions\*\*\*](#) and indicates CRC length. Value can be either one of CRC\_POLYLENGTH\_32B (32-bit CRC) CRC\_POLYLENGTH\_16B (16-bit CRC) CRC\_POLYLENGTH\_8B (8-bit CRC) CRC\_POLYLENGTH\_7B (7-bit CRC)
- ***uint32\_t CRC\_InitTypeDef::InitValue*** Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT\_INIT\_VALUE\_ENABLE
- ***uint32\_t CRC\_InitTypeDef::InputDataInversionMode*** This parameter is a value of [\*\*\*CRCEx\\_Input\\_Data\\_Inversion\*\*\*](#) and specifies input data inversion mode. Can be either one of the following values CRC\_INPUTDATA\_INVERSION\_NONE no input data inversion CRC\_INPUTDATA\_INVERSION\_BYTEx wise inversion, 0x1A2B3C4D becomes 0x58D43CB2 CRC\_INPUTDATA\_INVERSION\_HALFWORD halfword-wise inversion, 0x1A2B3C4D becomes 0xD458B23C CRC\_INPUTDATA\_INVERSION\_WORD word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458

- ***uint32\_t CRC\_InitTypeDef::OutputDataInversionMode*** This parameter is a value of **CRCEx\_Output\_Data\_Inversion** and specifies output data (i.e. CRC) inversion mode. Can be either CRC\_OUTPUTDATA\_INVERSION\_DISABLED no CRC inversion, or CRC\_OUTPUTDATA\_INVERSION\_ENABLED CRC 0x11223344 is converted into 0x22CC4488

### 10.1.2 CRC\_HandleTypeDef

**CRC\_HandleTypeDef** is defined in the `stm32f0xx_hal_crc.h`

#### Data Fields

- ***CRC\_TypeDef \* Instance***
- ***CRC\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_IO HAL\_CRC\_StateTypeDef State***
- ***uint32\_t InputDataFormat***

#### Field Documentation

- ***CRC\_TypeDef\* CRC\_HandleTypeDef::Instance*** Register base address
- ***CRC\_InitTypeDef CRC\_HandleTypeDef::Init*** CRC configuration parameters
- ***HAL\_LockTypeDef CRC\_HandleTypeDef::Lock*** CRC Locking object
- ***\_IO HAL\_CRC\_StateTypeDef CRC\_HandleTypeDef::State*** CRC communication state
- ***uint32\_t CRC\_HandleTypeDef::InputDataFormat*** This parameter is a value of **CRC\_Input\_Buffer\_Format** and specifies input data format. Can be either CRC\_INPUTDATA\_FORMAT\_BYTES input data is a stream of bytes (8-bit data) CRC\_INPUTDATA\_FORMAT\_HALFWORDS input data is a stream of half-words (16-bit data) CRC\_INPUTDATA\_FORMAT\_WORDS input data is a stream of words (32-bits data) Note that constant CRC\_INPUT\_FORMAT\_UNDEFINED is defined but an initialization error must occur if InputBufferFormat is not one of the three values listed above

## 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

1. Enable CRC AHB clock using `__CRC_CLK_ENABLE()`;
2. Initialize CRC calculator
  - specify generating polynomial (IP default or non-default one)
  - specify initialization value (IP default or non-default one)
  - specify input data format
  - specify input or output data inversion mode if any
3. Use `HAL_CRC_Accumulate()` function to compute the CRC value of the input data buffer starting with the previously computed CRC as initialization value

4. Use HAL\_CRC\_Calculate() function to compute the CRC value of the input data buffer starting with the defined initialization value (default or non-default) to initiate CRC calculation

### 10.2.2 Initialization and Configuration 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 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer using combination of the previous CRC value and the new one. or
- Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer independently of the previous CRC value.
- [\*\*HAL\\_CRC\\_Accumulate\(\)\*\*](#)
- [\*\*HAL\\_CRC\\_Calculate\(\)\*\*](#)

### 10.2.4 Peripheral State functions

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

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

### 10.2.5 HAL\_CRC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_CRC_Init ( <a href="#"><b>CRC_HandleTypeDef</b></a> * <b>hcrc</b>)</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> : CRC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.6 HAL\_CRC\_DeInit

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

### 10.2.7 HAL\_CRC\_MspInit

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

### 10.2.8 HAL\_CRC\_MspDeInit

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

### 10.2.9 HAL\_CRC\_Accumulate

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

### 10.2.10 HAL\_CRC\_Calculate

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

### 10.2.11 HAL\_CRC\_GetState

Function Name	<code>HAL_CRC_StateTypeDef HAL_CRC_GetState (</code>
---------------	------------------------------------------------------

***CRC\_HandleTypeDefDef \* hcrc)***

Function Description	Returns the CRC state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc</b> : CRC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 10.3 CRC Firmware driver defines

### 10.3.1 CRC

CRC

*Aliases for inter STM32 series compatibility*

HAL\_CRC\_Input\_Data\_Reverse

HAL\_CRC\_Output\_Data\_Reverse

*Default CRC computation initialization value*

DEFAULT\_CRC\_INITVALUE

*Indicates whether or not default init value is used*

DEFAULT\_INIT\_VALUE\_ENABLE

DEFAULT\_INIT\_VALUE\_DISABLE

IS\_DEFAULT\_INIT\_VALUE

*Indicates whether or not default polynomial is used*

DEFAULT\_POLYNOMIAL\_ENABLE

DEFAULT\_POLYNOMIAL\_DISABLE

IS\_DEFAULT\_POLYNOMIAL

*Default CRC generating polynomial*

DEFAULT\_CRC32\_POLY

**CRC Exported Macros**\_HAL\_CRC\_RESET\_HANDLE\_STATE**Description:**

- Reset CRC handle state.

**Parameters:**

- \_HANDLE\_: CRC handle.

**Return value:**

- None:

\_HAL\_CRC\_DR\_RESET**Description:**

- Reset CRC Data Register.

**Parameters:**

- `__HANDLE__`: CRC handle

**Return value:**

- None.:

**`__HAL_CRC_INITIALCRCVALUE_CONFIG`**

- Set CRC INIT non-default value.

**Parameters:**

- `__HANDLE__`: CRC handle
- `__INIT__`: 32-bit initial value

**Return value:**

- None.:

***Input Buffer Format***

`CRC_INPUTDATA_FORMAT_UNDEFINED`

`CRC_INPUTDATA_FORMAT_BYTES`

`CRC_INPUTDATA_FORMAT_HALFWORDS`

`CRC_INPUTDATA_FORMAT_WORDS`

`IS_CRC_INPUTDATA_FORMAT`

## 11 HAL CRC Extension Driver

### 11.1 CRCEEx Firmware driver API description

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

#### 11.1.1 Product specific features

#### 11.1.2 How to use this driver

- Extended initialization
- Set or not user-defined generating polynomial other than default one

#### 11.1.3 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize the CRC generating polynomial: if programmable polynomial feature is applicable to device, set default or non-default generating polynomial according to hcrc->Init.DefaultPolynomialUse parameter. If feature is non-applicable to device in use, HAL\_CRCEEx\_Init straight away reports HAL\_OK.
- Set the generating polynomial
- [`HAL\_CRCEEx\_Init\(\)`](#)
- [`HAL\_CRCEEx\_Input\_Data\_Reverse\(\)`](#)
- [`HAL\_CRCEEx\_Output\_Data\_Reverse\(\)`](#)
- [`HAL\_CRCEEx\_Polynomial\_Set\(\)`](#)

#### 11.1.4 HAL\_CRCEEx\_Init

Function Name	<code>HAL_StatusTypeDef HAL_CRCEEx_Init ( CRC_HandleTypeDef * hcrc)</code>
Function Description	Extended initialization to set generating polynomial.
Parameters	<ul style="list-style-type: none"><li>• <b>hcrc</b> : CRC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 11.1.5 HAL\_CRCEEx\_Input\_Data\_Reverse

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

### 11.1.6 HAL\_CRCEx\_Output\_Data\_Reverse

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

### 11.1.7 HAL\_CRCEx\_Polynomial\_Set

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

## 11.2 CRCEEx Firmware driver defines

### 11.2.1 CRCEEx

CRCEEx

#### *CRCEEx Exported Macros*

<code>_HAL_CRC_OUTPUTREVERSAL_ENABLE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• Set CRC output reversal.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: : CRC handle</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>• None.:</li> </ul>
<code>_HAL_CRC_OUTPUTREVERSAL_DISABLE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• Unset CRC output reversal.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: : CRC handle</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>• None.:</li> </ul>

[\\_\\_HAL\\_CRC\\_POLYNOMIAL\\_CONFIG](#)**Description:**

- Set CRC non-default polynomial.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): : CRC handle
- [\\_\\_POLYNOMIAL\\_\\_](#): 7, 8, 16 or 32-bit polynomial

**Return value:**

- None.:

***Input Data Inversion Modes***

[CRC\\_INPUTDATA\\_INVERSION\\_NONE](#)  
[CRC\\_INPUTDATA\\_INVERSION\\_BYTE](#)  
[CRC\\_INPUTDATA\\_INVERSION\\_HALFWORD](#)  
[CRC\\_INPUTDATA\\_INVERSION\\_WORD](#)  
[IS\\_CRC\\_INPUTDATA\\_INVERSION\\_MODE](#)

***Output Data Inversion Modes***

[CRC\\_OUTPUTDATA\\_INVERSION\\_DISABLED](#)  
[CRC\\_OUTPUTDATA\\_INVERSION\\_ENABLED](#)  
[IS\\_CRC\\_OUTPUTDATA\\_INVERSION\\_MODE](#)

***Polynomial sizes to configure the IP***

[CRC\\_POLYLENGTH\\_32B](#)  
[CRC\\_POLYLENGTH\\_16B](#)  
[CRC\\_POLYLENGTH\\_8B](#)  
[CRC\\_POLYLENGTH\\_7B](#)  
[IS\\_CRC\\_POL\\_LENGTH](#)

***CRC polynomial possible sizes actual definitions***

[HAL\\_CRC\\_LENGTH\\_32B](#)  
[HAL\\_CRC\\_LENGTH\\_16B](#)  
[HAL\\_CRC\\_LENGTH\\_8B](#)  
[HAL\\_CRC\\_LENGTH\\_7B](#)

## 12 HAL DAC Generic Driver

### 12.1 DAC Firmware driver registers structures

#### 12.1.1 DAC\_HandleTypeDef

*DAC\_HandleTypeDef* is defined in the `stm32f0xx_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

#### 12.1.2 DAC\_ChannelConfTypeDef

*DAC\_ChannelConfTypeDef* is defined in the `stm32f0xx_hal_dac.h`

##### Data Fields

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

##### Field Documentation

- *uint32\_t DAC\_ChannelConfTypeDef::DAC\_Trigger* Specifies the external trigger for the selected DAC channel. This parameter can be a value of [DAC\\_trigger\\_selection](#)
- *uint32\_t DAC\_ChannelConfTypeDef::DAC\_OutputBuffer* Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of [DAC\\_output\\_buffer](#)

### 12.2 DAC Firmware driver API description

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

## 12.2.1 DAC Peripheral features

### DAC Channels

STM32F0 devices integrates no, one or two 12-bit Digital Analog Converters. STM32F05x devices have one converter (channel1) STM32F07x & STM32F09x devices have two converters (i.e. channel1 & channel2) When 2 converters are present (i.e. channel1 & channel2) they can be used independently or simultaneously (dual mode):

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

### DAC Triggers

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

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx\_Pin9) using DAC\_Trigger\_Ext\_IT9. The used pin (GPIOx\_Pin9) must be configured in input mode.
2. Timers TRGO: TIM2, TIM3, TIM6, and TIM15 (DAC\_Trigger\_T2\_TRGO, DAC\_Trigger\_T3\_TRGO...)
3. Software using DAC\_Trigger\_Software

### DAC Buffer mode feature

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



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

### DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave
2. Triangle wave

### DAC data format

The DAC data format can be:

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

### DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation:  $DAC\_OUTx = VREF+ * DOR / 4095$  with DOR is the Data Output Register VEF+

is the input voltage reference (refer to the device datasheet) e.g. To set DAC\_OUT1 to 0.7V, use Assuming that VREF+ = 3.3V,  $DAC\_OUT1 = (3.3 * 868) / 4095 = 0.7V$

### DMA requests

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

DMA1 requests are mapped as following:

1. DAC channel1 : mapped on DMA1 channel3 which must be already configured
2. DAC channel2 : mapped on DMA1 channel4 which must be already configured For Dual mode and specific signal (Triangle and noise) generation please refer to Extension Features Driver description STM32F0 devices with one channel (one converting capability) does not support Dual mode and specific signal (Triangle and noise) generation.

## 12.2.2 How to use this driver

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

### Polling mode IO operation

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

### DMA mode IO operation

- Start the DAC peripheral using HAL\_DAC\_Start\_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion
- At The end of data transfer HAL\_DAC\_ConvCpltCallbackCh1() or HAL\_DAC\_ConvCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL\_DAC\_ConvCpltCallbackCh1 or HAL\_DAC\_ConvCpltCallbackCh2
- In case of transfer Error, HAL\_DAC\_ErrorCallbackCh1() function is executed and user can add his own code by customization of function pointer HAL\_DAC\_ErrorCallbackCh1
- Stop the DAC peripheral using HAL\_DAC\_Stop\_DMA()

### DAC HAL driver macros list

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

- \_\_HAL\_DAC\_ENABLE : Enable the DAC peripheral
- \_\_HAL\_DAC\_DISABLE : Disable the DAC peripheral
- \_\_HAL\_DAC\_CLEAR\_FLAG: Clear the DAC's pending flags

- `_HAL_DAC_GET_FLAG`: Get the selected DAC's flag status



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

### 12.2.3 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()`

### 12.2.4 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_ConvCpltCallbackCh1()`
- `HAL_DAC_ConvHalfCpltCallbackCh1()`
- `HAL_DAC_ErrorCallbackCh1()`
- `HAL_DAC_DMAUnderrunCallbackCh1()`

### 12.2.5 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()`
- `HAL_DAC_GetValue()`

### 12.2.6 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_IRQHandler()`

### 12.2.7 HAL\_DAC\_Init

Function Name	<code>HAL_StatusTypeDef HAL_DAC_Init ( DAC_HandleTypeDef * hdac)</code>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.8 HAL\_DAC\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_DAC_DeInit ( DAC_HandleTypeDef * hdac)</code>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.9 HAL\_DAC\_MspInit

Function Name	<code>void HAL_DAC_MspInit ( DAC_HandleTypeDef * hdac)</code>
Function Description	Initializes the DAC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that</li> </ul>

contains the configuration information for the specified DAC.

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

### 12.2.10 HAL\_DAC\_MspDeInit

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

### 12.2.11 HAL\_DAC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Start ( <i>DAC_HandleTypeDef</i> * hdac, <i>uint32_t</i> 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 <i>DAC_HandleTypeDef</i> 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:<ul style="list-style-type: none"><li>- <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li><li>- <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li><b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 12.2.12 HAL\_DAC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Stop ( <i>DAC_HandleTypeDef</i> * <i>hdac</i>, <i>uint32_t</i> <i>Channel</i>)</b>
Function Description	Disables DAC and stop 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: <ul style="list-style-type: none"> <li>– <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li> <li>– <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.13 HAL\_DAC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Start_DMA ( <i>DAC_HandleTypeDef</i> * <i>hdac</i>, <i>uint32_t</i> <i>Channel</i>, <i>uint32_t</i> * <i>pData</i>, <i>uint32_t</i> <i>Length</i>, <i>uint32_t</i> <i>Alignment</i>)</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: <ul style="list-style-type: none"> <li>– <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li> <li>– <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li> </ul> </li> <li>• <b>pData</b> : The destination peripheral Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from memory to DAC peripheral</li> <li>• <b>Alignment</b> : Specifies the data alignment for DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>DAC_ALIGN_8B_R</b> 8bit right data alignment selected</li> <li>– <b>DAC_ALIGN_12B_L</b> 12bit left data alignment selected</li> <li>– <b>DAC_ALIGN_12B_R</b> 12bit right data alignment selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.14 HAL\_DAC\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Stop_DMA (</b> <b>DAC_HandleTypeDef * hdac, uint32_t Channel)</b>
Function Description	Disables DAC and stop 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: <ul style="list-style-type: none"> <li>– <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li> <li>– <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.15 HAL\_DAC\_ConvCpltCallbackCh1

Function Name	<b>void HAL_DAC_ConvCpltCallbackCh1 (</b> <b>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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.16 HAL\_DAC\_ConvHalfCpltCallbackCh1

Function Name	<b>void HAL_DAC_ConvHalfCpltCallbackCh1 (</b> <b>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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.17 HAL\_DAC\_ErrorCallbackCh1

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

### 12.2.18 HAL\_DAC\_DMADebugCallbackCh1

Function Name	<code>void HAL_DAC_DMADebugCallbackCh1 ( <i>DAC_HandleTypeDef</i> * hdac)</code>
Function Description	DMA debug DAC callback for channel1.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.19 HAL\_DAC\_ConfigChannel

Function Name	<code>HAL_StatusTypeDef HAL_DAC_ConfigChannel ( <i>DAC_HandleTypeDef</i> * hdac, <i>DAC_ChannelConfTypeDef</i> * sConfig, uint32_t Channel)</code>
Function Description	Configures the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li> </ul>

---

	<ul style="list-style-type: none"> <li>• <b>sConfig</b> : DAC configuration structure.</li> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li> <li>– <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li> </ul> </li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

## 12.2.20 HAL\_DAC\_SetValue

Function Name	<code>HAL_StatusTypeDef HAL_DAC_SetValue (</code> <code>DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t</code> <code>Alignment, uint32_t Data)</code>
Function Description	Set the specified data holding register value for DAC channel.
Parameters	<ul style="list-style-type: none"> <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:           <ul style="list-style-type: none"> <li>– <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li> <li>– <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li> </ul> </li> <li>• <b>Alignment</b> : Specifies the data alignment. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b>DAC_ALIGN_8B_R</b> 8bit right data alignment selected</li> <li>– <b>DAC_ALIGN_12B_L</b> 12bit left data alignment selected</li> <li>– <b>DAC_ALIGN_12B_R</b> 12bit right data alignment selected</li> </ul> </li> <li>• <b>Data</b> : Data to be loaded in the selected data holding register.</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

## 12.2.21 HAL\_DAC\_GetValue

Function Name	<code>uint32_t HAL_DAC_GetValue ( DAC_HandleTypeDef * hdac,</code> <code>uint32_t Channel)</code>
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> <li><b>Channel</b> : The selected DAC channel. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li> <li>– <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>The selected DAC channel data output value.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 12.2.22 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><b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 12.2.23 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><b>DAC Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 12.2.24 HAL\_DAC\_IRQHandler

Function Name	<b>void HAL_DAC_IRQHandler ( <i>DAC_HandleTypeDef</i> * hdac)</b>
Function Description	Handles DAC interrupt request.
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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 12.3 DAC Firmware driver defines

### 12.3.1 DAC

DAC

***DAC Channel selection***

DAC\_CHANNEL\_1

DAC\_CHANNEL\_2

IS\_DAC\_CHANNEL

***DAC data***

IS\_DAC\_DATA

***DAC data alignment***

DAC\_ALIGN\_12B\_R

DAC\_ALIGN\_12B\_L

DAC\_ALIGN\_8B\_R

IS\_DAC\_ALIGN

***DAC Error Code***

HAL_DAC_ERROR_NONE	No error
--------------------	----------

HAL_DAC_ERROR_DMAUNDERUNCH1	DAC channel1 DAM underrun error
-----------------------------	---------------------------------

HAL_DAC_ERROR_DMAUNDERUNCH2	DAC channel2 DAM underrun error
-----------------------------	---------------------------------

HAL_DAC_ERROR_DMA	DMA error
-------------------	-----------

***DAC Exported Macros***

<u>_HAL_DAC_RESET_HANDLE_STATE</u>	<b>Description:</b>
------------------------------------	---------------------

- 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:

`__HAL_DAC_DISABLE`

**Description:**

- Disable the DAC channel.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle
- `__DAC_Channel__`: specifies the DAC channel.

**Return value:**

- None:

`__HAL_DHR12R1_ALIGNEMENT`

**Description:**

- Set DHR12R1 alignment.

**Parameters:**

- `__ALIGNEMENT__`: specifies the DAC alignment

**Return value:**

- None:

`__HAL_DHR12R2_ALIGNEMENT`

**Description:**

- Set DHR12R2 alignment.

**Parameters:**

- `__ALIGNEMENT__`: specifies the DAC alignment

**Return value:**

- None:

`__HAL_DHR12RD_ALIGNEMENT`

**Description:**

- Set DHR12RD alignment.

**Parameters:**

- `__ALIGNEMENT__`: specifies the DAC alignment

**Return value:**

- None:

<code>__HAL_DAC_ENABLE_IT</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Enable the DAC interrupt.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li><code>__HANDLE__</code>: specifies the DAC handle</li> <li><code>__INTERRUPT__</code>: specifies the DAC interrupt.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None:</li> </ul>
<code>__HAL_DAC_DISABLE_IT</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Disable the DAC interrupt.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li><code>__HANDLE__</code>: specifies the DAC handle</li> <li><code>__INTERRUPT__</code>: specifies the DAC interrupt.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None:</li> </ul>
<code>__HAL_DAC_GET_FLAG</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Get the selected DAC's flag status.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li><code>__HANDLE__</code>: specifies the DAC handle.</li> <li><code>__FLAG__</code>: specifies the FLAG.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None:</li> </ul>
<code>__HAL_DAC_CLEAR_FLAG</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Clear the DAC's flag.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li><code>__HANDLE__</code>: specifies the DAC handle.</li> <li><code>__FLAG__</code>: specifies the FLAG.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None:</li> </ul>

***DAC flags definition***`DAC_FLAG_DMAUDR1``DAC_FLAG_DMAUDR2`***DAC IT definition***`DAC_IT_DMAUDR1``DAC_IT_DMAUDR2`***DAC output buffer***`DAC_OUTPUTBUFFER_ENABLE`

DAC\_OUTPUTBUFFER\_DISABLE  
IS\_DAC\_OUTPUT\_BUFFER\_STATE

***DAC trigger selection***

DAC_TRIGGER_NONE	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
DAC_TRIGGER_T2_TRGO	TIM2 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T3_TRGO	TIM3 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T6_TRGO	TIM6 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T7_TRGO	TIM7 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T15_TRGO	TIM15 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
IS_DAC_TRIGGER	

## 13 HAL DAC Extension Driver

### 13.1 DACEx Firmware driver API description

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

#### 13.1.1 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.

#### 13.1.2 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\_DAC\_ConfigChannel\(\)`](#)
- [`HAL\_DAC\_GetValue\(\)`](#)
- [`HAL\_DAC\_Start\(\)`](#)
- [`HAL\_DAC\_Start\_DMA\(\)`](#)
- [`HAL\_DAC\_IRQHandler\(\)`](#)
- [`HAL\_DACEx\_DualGetValue\(\)`](#)
- [`HAL\_DACEx\_TriangleWaveGenerate\(\)`](#)
- [`HAL\_DACEx\_NoiseWaveGenerate\(\)`](#)
- [`HAL\_DACEx\_DualSetValue\(\)`](#)
- [`HAL\_DACEx\_ConvCpltCallbackCh2\(\)`](#)
- [`HAL\_DACEx\_ConvHalfCpltCallbackCh2\(\)`](#)
- [`HAL\_DACEx\_ErrorCallbackCh2\(\)`](#)
- [`HAL\_DACEx\_DMAUnderrunCallbackCh2\(\)`](#)
- [`DAC\_DMAConvCpltCh2\(\)`](#)
- [`DAC\_DMAMhalfConvCpltCh2\(\)`](#)
- [`DAC\_DMAErrorCh2\(\)`](#)

#### 13.1.3 `HAL_DAC_ConfigChannel`

Function Name

`HAL_StatusTypeDef HAL_DAC_ConfigChannel (`  
[`DAC\_HandleTypeDef \* hdac,`](#) [`DAC\_ChannelConfTypeDef \*`](#)

	<b>sConfig, uint32_t Channel)</b>
Function Description	Configures 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> <li>• <b>sConfig</b> : DAC configuration structure.</li> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li> <li>- <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.4 HAL\_DAC\_GetValue

Function Name	<b>uint32_t HAL_DAC_GetValue ( DAC_HandleTypeDef * hdac, uint32_t Channel)</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> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li> <li>- <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>The selected DAC channel data output value.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.5 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:</li> </ul>

	<ul style="list-style-type: none"> <li>- <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li> <li>- <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.6 HAL\_DAC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Start_DMA (</b> <b>DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)</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: <ul style="list-style-type: none"> <li>- <b>DAC_CHANNEL_1</b> DAC Channel1 selected</li> <li>- <b>DAC_CHANNEL_2</b> DAC Channel2 selected</li> </ul> </li> <li>• <b>pData</b> : The destination peripheral Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from memory to DAC peripheral</li> <li>• <b>Alignment</b> : Specifies the data alignment for DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>DAC_ALIGN_8B_R</b> 8bit right data alignment selected</li> <li>- <b>DAC_ALIGN_12B_L</b> 12bit left data alignment selected</li> <li>- <b>DAC_ALIGN_12B_R</b> 12bit right data alignment selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.7 HAL\_DAC\_IRQHandler

Function Name	<b>void HAL_DAC_IRQHandler (</b> <b>DAC_HandleTypeDef * hdac)</b>
Function Description	Handles DAC interrupt request.
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>

## Notes

- None.

### 13.1.8 HAL\_DACEx\_DualGetValue

Function Name	<code>uint32_t HAL_DACEx_DualGetValue ( <i>DAC_HandleTypeDef</i> * hdac)</code>
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 <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>The selected DAC channel data output value.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.9 HAL\_DACEx\_TriangleWaveGenerate

Function Name	<code>HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate ( <i>DAC_HandleTypeDef</i> * hdac, uint32_t Channel, uint32_t Amplitude)</code>
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 <i>DAC_HandleTypeDef</i> 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: <i>DAC_CHANNEL_1</i> / <i>DAC_CHANNEL_2</i></li> <li>• <b>Amplitude</b> : Select max triangle amplitude. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>DAC_TRIANGLEAMPLITUDE_1</i></b> Select max triangle amplitude of 1</li> <li>– <b><i>DAC_TRIANGLEAMPLITUDE_3</i></b> Select max triangle amplitude of 3</li> <li>– <b><i>DAC_TRIANGLEAMPLITUDE_7</i></b> Select max triangle amplitude of 7</li> <li>– <b><i>DAC_TRIANGLEAMPLITUDE_15</i></b> Select max triangle amplitude of 15</li> <li>– <b><i>DAC_TRIANGLEAMPLITUDE_31</i></b> Select max triangle amplitude of 31</li> <li>– <b><i>DAC_TRIANGLEAMPLITUDE_63</i></b> Select max triangle amplitude of 63</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- <b>DAC_TRIANGLEAMPLITUDE_127</b> Select max triangle amplitude of 127</li> <li>- <b>DAC_TRIANGLEAMPLITUDE_255</b> Select max triangle amplitude of 255</li> <li>- <b>DAC_TRIANGLEAMPLITUDE_511</b> Select max triangle amplitude of 511</li> <li>- <b>DAC_TRIANGLEAMPLITUDE_1023</b> Select max triangle amplitude of 1023</li> <li>- <b>DAC_TRIANGLEAMPLITUDE_2047</b> Select max triangle amplitude of 2047</li> <li>- <b>DAC_TRIANGLEAMPLITUDE_4095</b> Select max triangle amplitude of 4095</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.10 HAL\_DACEx\_NoiseWaveGenerate

Function Name	<b>HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (</b> <b>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: <ul style="list-style-type: none"> <li>- <b>DAC_LFSRUNMASK_BIT0</b> Unmask DAC channel LFSR bit0 for noise wave generation</li> <li>- <b>DAC_LFSRUNMASK_BITS1_0</b> Unmask DAC channel LFSR bit[1:0] for noise wave generation</li> <li>- <b>DAC_LFSRUNMASK_BITS2_0</b> Unmask DAC channel LFSR bit[2:0] for noise wave generation</li> <li>- <b>DAC_LFSRUNMASK_BITS3_0</b> Unmask DAC channel LFSR bit[3:0] for noise wave generation</li> <li>- <b>DAC_LFSRUNMASK_BITS4_0</b> Unmask DAC channel LFSR bit[4:0] for noise wave generation</li> <li>- <b>DAC_LFSRUNMASK_BITS5_0</b> Unmask DAC channel LFSR bit[5:0] for noise wave generation</li> <li>- <b>DAC_LFSRUNMASK_BITS6_0</b> Unmask DAC channel LFSR bit[6:0] for noise wave generation</li> <li>- <b>DAC_LFSRUNMASK_BITS7_0</b> Unmask DAC channel LFSR bit[7:0] for noise wave generation</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b>DAC_LFSRUNMASK_BITS8_0</b> Unmask DAC channel LFSR bit[8:0] for noise wave generation</li> <li>- <b>DAC_LFSRUNMASK_BITS9_0</b> Unmask DAC channel LFSR bit[9:0] for noise wave generation</li> <li>- <b>DAC_LFSRUNMASK_BITS10_0</b> Unmask DAC channel LFSR bit[10:0] for noise wave generation</li> <li>- <b>DAC_LFSRUNMASK_BITS11_0</b> Unmask DAC channel LFSR bit[11:0] for noise wave generation</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.11 HAL\_DACEx\_DualSetValue

Function Name	<b>HAL_StatusTypeDef HAL_DACEx_DualSetValue (</b> <b>DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)</b>
Function Description	Set the specified data holding register value for dual 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> <li>• <b>Alignment</b> : 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</li> <li>• <b>Data1</b> : Data for DAC Channel2 to be loaded in the selected data holding register.</li> <li>• <b>Data2</b> : Data for DAC Channel1 to be loaded in the selected data holding register.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In dual mode, a unique register access is required to write in both DAC channels at the same time.</li> </ul>

### 13.1.12 HAL\_DACEx\_ConvCpltCallbackCh2

Function Name	<b>void HAL_DACEx_ConvCpltCallbackCh2 (</b> <b>DAC_HandleTypeDef * hdac)</b>
---------------	---------------------------------------------------------------------------------

Function Description	Conversion complete callback in non blocking mode for Channel2.
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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 13.1.13 HAL\_DACEx\_ConvHalfCpltCallbackCh2

Function Name	<b>void HAL_DACEx_ConvHalfCpltCallbackCh2 ( DAC_HandleTypeDef * hdac)</b>
Function Description	Conversion half DMA transfer callback in non blocking mode for Channel2.
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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 13.1.14 HAL\_DACEx\_ErrorCallbackCh2

Function Name	<b>void HAL_DACEx_ErrorCallbackCh2 ( DAC_HandleTypeDef * hdac)</b>
Function Description	Error DAC callback for Channel2.
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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 13.1.15 HAL\_DACEx\_DMADebugCallbackCh2

---

Function Name	<b>void HAL_DACEx_DMAUnderrunCallbackCh2 ( DAC_HandleTypeDef * hdac)</b>
Function Description	DMA underrun DAC callback for channel2.
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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.16 DAC\_DMAConvCpltCh2

Function Name	<b>void DAC_DMAConvCpltCh2 ( DMA_HandleTypeDef * hdma)</b>
Function Description	DMA conversion complete 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.17 DAC\_DMALeftConvCpltCh2

Function Name	<b>void DAC_DMALeftConvCpltCh2 ( DMA_HandleTypeDef * hdma)</b>
Function Description	DMA half transfer complete 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.18 DAC\_DMAErrorCh2

Function Name	<b>void DAC_DMAErrorCh2 ( DMA_HandleTypeDef * hdma)</b>
Function Description	DMA 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 13.2 DACEx Firmware driver defines

### 13.2.1 DACEx

DACEx

*DACEx Ifsrnunmask 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
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

DAC_TRIANGLEAMPLITUDE_1	bit[11:0] for noise wave generation
DAC_TRIANGLEAMPLITUDE_3	Select max triangle amplitude of 1
DAC_TRIANGLEAMPLITUDE_7	Select max triangle amplitude of 3
DAC_TRIANGLEAMPLITUDE_15	Select max triangle amplitude of 7
DAC_TRIANGLEAMPLITUDE_31	Select max triangle amplitude of 15
DAC_TRIANGLEAMPLITUDE_63	Select max triangle amplitude of 31
DAC_TRIANGLEAMPLITUDE_127	Select max triangle amplitude of 63
DAC_TRIANGLEAMPLITUDE_255	Select max triangle amplitude of 127
DAC_TRIANGLEAMPLITUDE_511	Select max triangle amplitude of 255
DAC_TRIANGLEAMPLITUDE_1023	Select max triangle amplitude of 511
DAC_TRIANGLEAMPLITUDE_2047	Select max triangle amplitude of 1023
DAC_TRIANGLEAMPLITUDE_4095	Select max triangle amplitude of 2047
IS_DAC_LFSR_UNMASK_TRIANGLE_AMPLITUDE	Select max triangle amplitude of 4095
<b>DACEx wave generation</b>	
DAC_WAVEGENERATION_NONE	
DAC_WAVEGENERATION_NOISE	
DAC_WAVEGENERATION_TRIANGLE	
IS_DAC_GENERATE_WAVE	
<b>DACEx wave generation bis</b>	
DAC_WAVE_NOISE	
DAC_WAVE_TRIANGLE	
IS_DAC_WAVE	

## 14 HAL DMA Generic Driver

### 14.1 DMA Firmware driver registers structures

#### 14.1.1 DMA\_InitTypeDef

*DMA\_InitTypeDef* is defined in the `stm32f0xx_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 DMA<sub>Y</sub> Channel<sub>X</sub>. 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 DMA<sub>Y</sub> Channel<sub>X</sub>. This parameter can be a value of [\*DMA\\_Priority\\_level\*](#)

#### 14.1.2 \_\_DMA\_HandleTypeDef

*\_\_DMA\_HandleTypeDef* is defined in the `stm32f0xx_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

## 14.2 DMA Firmware driver API description

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

### 14.2.1 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

### 14.2.2 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\(\)\*\*](#)

### 14.2.3 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\(\)\*\*](#)

#### 14.2.4 State and Errors functions

This subsection provides functions allowing to

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

#### 14.2.5 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 14.2.6 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 14.2.7 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><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 14.2.8 HAL\_DMA\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Start_IT ( DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)</b>
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><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 14.2.9 HAL\_DMA\_Abort

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Abort ( DMA_HandleTypeDef * hdma)</b>
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><b>HAL status</b></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>

#### 14.2.10 HAL\_DMA\_PollForTransfer

Function Name	<code>HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, uint32_t CompleteLevel, uint32_t Timeout)</code>
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><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 14.2.11 HAL\_DMA\_IRQHandler

Function Name	<code>void HAL_DMA_IRQHandler ( DMA_HandleTypeDef * hdma )</code>
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>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 14.2.12 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>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 14.2.13 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>• <b>DMA Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 14.3 DMA Firmware driver defines

### 14.3.1 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 Code**

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:**

- None:

`_HAL_DMA_DISABLE_IT`

**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	
<b>DMA Peripheral data size</b>	
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	
<b>DMA Peripheral incremented mode</b>	
DMA_PINC_ENABLE	Peripheral increment mode Enable
DMA_PINC_DISABLE	Peripheral increment mode Disable
IS_DMA_PERIPHERAL_INC_STATE	
<b>DMA Priority level</b>	
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	
<b>DMA Private Constants</b>	
HAL_TIMEOUT_DMA_ABORT	

## 15 HAL DMA Extension Driver

### 15.1 DMAEx Firmware driver defines

#### 15.1.1 DMAEx

DMAEx

**DMAEx Exported Constants**

DMA1\_CHANNEL1\_RMP  
DMA1\_CHANNEL2\_RMP  
DMA1\_CHANNEL3\_RMP  
DMA1\_CHANNEL4\_RMP  
DMA1\_CHANNEL5\_RMP  
DMA1\_CHANNEL6\_RMP  
DMA1\_CHANNEL7\_RMP  
DMA2\_CHANNEL1\_RMP  
DMA2\_CHANNEL2\_RMP  
DMA2\_CHANNEL3\_RMP  
DMA2\_CHANNEL4\_RMP  
DMA2\_CHANNEL5\_RMP

HAL\_DMA1\_CH1\_DEFAULT Default remap position for DMA1  
HAL\_DMA1\_CH1\_ADC Remap ADC on DMA1 Channel 1  
HAL\_DMA1\_CH1\_TIM17\_CH1 Remap TIM17 channel 1 on DMA1 channel 1  
HAL\_DMA1\_CH1\_TIM17\_UP Remap TIM17 up on DMA1 channel 1  
HAL\_DMA1\_CH1\_USART1\_RX Remap USART1 Rx on DMA1 channel 1  
HAL\_DMA1\_CH1\_USART2\_RX Remap USART2 Rx on DMA1 channel 1  
HAL\_DMA1\_CH1\_USART3\_RX Remap USART3 Rx on DMA1 channel 1  
HAL\_DMA1\_CH1\_USART4\_RX Remap USART4 Rx on DMA1 channel 1  
HAL\_DMA1\_CH1\_USART5\_RX Remap USART5 Rx on DMA1 channel 1  
HAL\_DMA1\_CH1\_USART6\_RX Remap USART6 Rx on DMA1 channel 1  
HAL\_DMA1\_CH1\_USART7\_RX Remap USART7 Rx on DMA1 channel 1  
HAL\_DMA1\_CH1\_USART8\_RX Remap USART8 Rx on DMA1 channel 1  
HAL\_DMA1\_CH2\_DEFAULT Default remap position for DMA1  
HAL\_DMA1\_CH2\_ADC Remap ADC on DMA1 channel 2  
HAL\_DMA1\_CH2\_I2C1\_TX Remap I2C1 Tx on DMA1 channel 2  
HAL\_DMA1\_CH2\_SPI1\_RX Remap SPI1 Rx on DMA1 channel 2  
HAL\_DMA1\_CH2\_TIM1\_CH1 Remap TIM1 channel 1 on DMA1 channel 2

HAL_DMA1_CH2_TIM17_CH1	Remap TIM17 channel 1 on DMA1 channel 2
HAL_DMA1_CH2_TIM17_UP	Remap TIM17 up on DMA1 channel 2
HAL_DMA1_CH2_USART1_TX	Remap USART1 Tx on DMA1 channel 2
HAL_DMA1_CH2_USART2_TX	Remap USART2 Tx on DMA1 channel 2
HAL_DMA1_CH2_USART3_TX	Remap USART3 Tx on DMA1 channel 2
HAL_DMA1_CH2_USART4_TX	Remap USART4 Tx on DMA1 channel 2
HAL_DMA1_CH2_USART5_TX	Remap USART5 Tx on DMA1 channel 2
HAL_DMA1_CH2_USART6_TX	Remap USART6 Tx on DMA1 channel 2
HAL_DMA1_CH2_USART7_TX	Remap USART7 Tx on DMA1 channel 2
HAL_DMA1_CH2_USART8_TX	Remap USART8 Tx on DMA1 channel 2
HAL_DMA1_CH3_DEFAULT	Default remap position for DMA1
HAL_DMA1_CH3_TIM6_UP	Remap TIM6 up on DMA1 channel 3
HAL_DMA1_CH3_DAC_CH1	Remap DAC Channel 1 on DMA1 channel 3
HAL_DMA1_CH3_I2C1_RX	Remap I2C1 Rx on DMA1 channel 3
HAL_DMA1_CH3_SPI1_TX	Remap SPI1 Tx on DMA1 channel 3
HAL_DMA1_CH3_TIM1_CH2	Remap TIM1 channel 2 on DMA1 channel 3
HAL_DMA1_CH3_TIM2_CH2	Remap TIM2 channel 2 on DMA1 channel 3
HAL_DMA1_CH3_TIM16_CH1	Remap TIM16 channel 1 on DMA1 channel 3
HAL_DMA1_CH3_TIM16_UP	Remap TIM16 up on DMA1 channel 3
HAL_DMA1_CH3_USART1_RX	Remap USART1 Rx on DMA1 channel 3
HAL_DMA1_CH3_USART2_RX	Remap USART2 Rx on DMA1 channel 3
HAL_DMA1_CH3_USART3_RX	Remap USART3 Rx on DMA1 channel 3
HAL_DMA1_CH3_USART4_RX	Remap USART4 Rx on DMA1 channel 3
HAL_DMA1_CH3_USART5_RX	Remap USART5 Rx on DMA1 channel 3
HAL_DMA1_CH3_USART6_RX	Remap USART6 Rx on DMA1 channel 3
HAL_DMA1_CH3_USART7_RX	Remap USART7 Rx on DMA1 channel 3
HAL_DMA1_CH3_USART8_RX	Remap USART8 Rx on DMA1 channel 3
HAL_DMA1_CH4_DEFAULT	Default remap position for DMA1
HAL_DMA1_CH4_TIM7_UP	Remap TIM7 up on DMA1 channel 4
HAL_DMA1_CH4_DAC_CH2	Remap DAC Channel 2 on DMA1 channel 4
HAL_DMA1_CH4_I2C2_TX	Remap I2C2 Tx on DMA1 channel 4
HAL_DMA1_CH4_SPI2_RX	Remap SPI2 Rx on DMA1 channel 4
HAL_DMA1_CH4_TIM2_CH4	Remap TIM2 channel 4 on DMA1 channel 4
HAL_DMA1_CH4_TIM3_CH1	Remap TIM3 channel 1 on DMA1 channel 4
HAL_DMA1_CH4_TIM3_TRIG	Remap TIM3 Trig on DMA1 channel 4
HAL_DMA1_CH4_TIM16_CH1	Remap TIM16 channel 1 on DMA1 channel 4

HAL_DMA1_CH4_TIM16_UP	Remap TIM16 up on DMA1 channel 4
HAL_DMA1_CH4_USART1_TX	Remap USART1 Tx on DMA1 channel 4
HAL_DMA1_CH4_USART2_TX	Remap USART2 Tx on DMA1 channel 4
HAL_DMA1_CH4_USART3_TX	Remap USART3 Tx on DMA1 channel 4
HAL_DMA1_CH4_USART4_TX	Remap USART4 Tx on DMA1 channel 4
HAL_DMA1_CH4_USART5_TX	Remap USART5 Tx on DMA1 channel 4
HAL_DMA1_CH4_USART6_TX	Remap USART6 Tx on DMA1 channel 4
HAL_DMA1_CH4_USART7_TX	Remap USART7 Tx on DMA1 channel 4
HAL_DMA1_CH4_USART8_TX	Remap USART8 Tx on DMA1 channel 4
HAL_DMA1_CH5_DEFAULT	Default remap position for DMA1
HAL_DMA1_CH5_I2C2_RX	Remap I2C2 Rx on DMA1 channel 5
HAL_DMA1_CH5_SPI2_TX	Remap SPI1 Tx on DMA1 channel 5
HAL_DMA1_CH5_TIM1_CH3	Remap TIM1 channel 3 on DMA1 channel 5
HAL_DMA1_CH5_USART1_RX	Remap USART1 Rx on DMA1 channel 5
HAL_DMA1_CH5_USART2_RX	Remap USART2 Rx on DMA1 channel 5
HAL_DMA1_CH5_USART3_RX	Remap USART3 Rx on DMA1 channel 5
HAL_DMA1_CH5_USART4_RX	Remap USART4 Rx on DMA1 channel 5
HAL_DMA1_CH5_USART5_RX	Remap USART5 Rx on DMA1 channel 5
HAL_DMA1_CH5_USART6_RX	Remap USART6 Rx on DMA1 channel 5
HAL_DMA1_CH5_USART7_RX	Remap USART7 Rx on DMA1 channel 5
HAL_DMA1_CH5_USART8_RX	Remap USART8 Rx on DMA1 channel 5
HAL_DMA1_CH6_DEFAULT	Default remap position for DMA1
HAL_DMA1_CH6_I2C1_TX	Remap I2C1 Tx on DMA1 channel 6
HAL_DMA1_CH6_SPI2_RX	Remap SPI2 Rx on DMA1 channel 6
HAL_DMA1_CH6_TIM1_CH1	Remap TIM1 channel 1 on DMA1 channel 6
HAL_DMA1_CH6_TIM1_CH2	Remap TIM1 channel 2 on DMA1 channel 6
HAL_DMA1_CH6_TIM1_CH3	Remap TIM1 channel 3 on DMA1 channel 6
HAL_DMA1_CH6_TIM3_CH1	Remap TIM3 channel 1 on DMA1 channel 6
HAL_DMA1_CH6_TIM3_TRIG	Remap TIM3 Trig on DMA1 channel 6
HAL_DMA1_CH6_TIM16_CH1	Remap TIM16 channel 1 on DMA1 channel 6
HAL_DMA1_CH6_TIM16_UP	Remap TIM16 up on DMA1 channel 6
HAL_DMA1_CH6_USART1_RX	Remap USART1 Rx on DMA1 channel 6
HAL_DMA1_CH6_USART2_RX	Remap USART2 Rx on DMA1 channel 6
HAL_DMA1_CH6_USART3_RX	Remap USART3 Rx on DMA1 channel 6
HAL_DMA1_CH6_USART4_RX	Remap USART4 Rx on DMA1 channel 6
HAL_DMA1_CH6_USART5_RX	Remap USART5 Rx on DMA1 channel 6

HAL_DMA1_CH6_USART6_RX	Remap USART6 Rx on DMA1 channel 6
HAL_DMA1_CH6_USART7_RX	Remap USART7 Rx on DMA1 channel 6
HAL_DMA1_CH6_USART8_RX	Remap USART8 Rx on DMA1 channel 6
HAL_DMA1_CH7_DEFAULT	Default remap position for DMA1
HAL_DMA1_CH7_I2C1_RX	Remap I2C1 Rx on DMA1 channel 7
HAL_DMA1_CH7_SPI2_TX	Remap SPI2 Tx on DMA1 channel 7
HAL_DMA1_CH7_TIM2_CH2	Remap TIM2 channel 2 on DMA1 channel 7
HAL_DMA1_CH7_TIM2_CH4	Remap TIM2 channel 4 on DMA1 channel 7
HAL_DMA1_CH7_TIM17_CH1	Remap TIM17 channel 1 on DMA1 channel 7
HAL_DMA1_CH7_TIM17_UP	Remap TIM17 up on DMA1 channel 7
HAL_DMA1_CH7_USART1_TX	Remap USART1 Tx on DMA1 channel 7
HAL_DMA1_CH7_USART2_TX	Remap USART2 Tx on DMA1 channel 7
HAL_DMA1_CH7_USART3_TX	Remap USART3 Tx on DMA1 channel 7
HAL_DMA1_CH7_USART4_TX	Remap USART4 Tx on DMA1 channel 7
HAL_DMA1_CH7_USART5_TX	Remap USART5 Tx on DMA1 channel 7
HAL_DMA1_CH7_USART6_TX	Remap USART6 Tx on DMA1 channel 7
HAL_DMA1_CH7_USART7_TX	Remap USART7 Tx on DMA1 channel 7
HAL_DMA1_CH7_USART8_TX	Remap USART8 Tx on DMA1 channel 7
HAL_DMA2_CH1_DEFAULT	Default remap position for DMA2
HAL_DMA2_CH1_I2C2_TX	Remap I2C2 TX on DMA2 channel 1
HAL_DMA2_CH1_USART1_TX	Remap USART1 Tx on DMA2 channel 1
HAL_DMA2_CH1_USART2_TX	Remap USART2 Tx on DMA2 channel 1
HAL_DMA2_CH1_USART3_TX	Remap USART3 Tx on DMA2 channel 1
HAL_DMA2_CH1_USART4_TX	Remap USART4 Tx on DMA2 channel 1
HAL_DMA2_CH1_USART5_TX	Remap USART5 Tx on DMA2 channel 1
HAL_DMA2_CH1_USART6_TX	Remap USART6 Tx on DMA2 channel 1
HAL_DMA2_CH1_USART7_TX	Remap USART7 Tx on DMA2 channel 1
HAL_DMA2_CH1_USART8_TX	Remap USART8 Tx on DMA2 channel 1
HAL_DMA2_CH2_DEFAULT	Default remap position for DMA2
HAL_DMA2_CH2_I2C2_RX	Remap I2C2 Rx on DMA2 channel 2
HAL_DMA2_CH2_USART1_RX	Remap USART1 Rx on DMA2 channel 2
HAL_DMA2_CH2_USART2_RX	Remap USART2 Rx on DMA2 channel 2
HAL_DMA2_CH2_USART3_RX	Remap USART3 Rx on DMA2 channel 2
HAL_DMA2_CH2_USART4_RX	Remap USART4 Rx on DMA2 channel 2
HAL_DMA2_CH2_USART5_RX	Remap USART5 Rx on DMA2 channel 2
HAL_DMA2_CH2_USART6_RX	Remap USART6 Rx on DMA2 channel 2

HAL_DMA2_CH2_USART7_RX	Remap USART7 Rx on DMA2 channel 2
HAL_DMA2_CH2_USART8_RX	Remap USART8 Rx on DMA2 channel 2
HAL_DMA2_CH3_DEFAULT	Default remap position for DMA2
HAL_DMA2_CH3_TIM6_UP	Remap TIM6 up on DMA2 channel 3
HAL_DMA2_CH3_DAC_CH1	Remap DAC channel 1 on DMA2 channel 3
HAL_DMA2_CH3_SPI1_RX	Remap SPI1 Rx on DMA2 channel 3
HAL_DMA2_CH3_USART1_RX	Remap USART1 Rx on DMA2 channel 3
HAL_DMA2_CH3_USART2_RX	Remap USART2 Rx on DMA2 channel 3
HAL_DMA2_CH3_USART3_RX	Remap USART3 Rx on DMA2 channel 3
HAL_DMA2_CH3_USART4_RX	Remap USART4 Rx on DMA2 channel 3
HAL_DMA2_CH3_USART5_RX	Remap USART5 Rx on DMA2 channel 3
HAL_DMA2_CH3_USART6_RX	Remap USART6 Rx on DMA2 channel 3
HAL_DMA2_CH3_USART7_RX	Remap USART7 Rx on DMA2 channel 3
HAL_DMA2_CH3_USART8_RX	Remap USART8 Rx on DMA2 channel 3
HAL_DMA2_CH4_DEFAULT	Default remap position for DMA2
HAL_DMA2_CH4_TIM7_UP	Remap TIM7 up on DMA2 channel 4
HAL_DMA2_CH4_DAC_CH2	Remap DAC channel 2 on DMA2 channel 4
HAL_DMA2_CH4_SPI1_TX	Remap SPI1 Tx on DMA2 channel 4
HAL_DMA2_CH4_USART1_TX	Remap USART1 Tx on DMA2 channel 4
HAL_DMA2_CH4_USART2_TX	Remap USART2 Tx on DMA2 channel 4
HAL_DMA2_CH4_USART3_TX	Remap USART3 Tx on DMA2 channel 4
HAL_DMA2_CH4_USART4_TX	Remap USART4 Tx on DMA2 channel 4
HAL_DMA2_CH4_USART5_TX	Remap USART5 Tx on DMA2 channel 4
HAL_DMA2_CH4_USART6_TX	Remap USART6 Tx on DMA2 channel 4
HAL_DMA2_CH4_USART7_TX	Remap USART7 Tx on DMA2 channel 4
HAL_DMA2_CH4_USART8_TX	Remap USART8 Tx on DMA2 channel 4
HAL_DMA2_CH5_DEFAULT	Default remap position for DMA2
HAL_DMA2_CH5_ADC	Remap ADC on DMA2 channel 5
HAL_DMA2_CH5_USART1_TX	Remap USART1 Tx on DMA2 channel 5
HAL_DMA2_CH5_USART2_TX	Remap USART2 Tx on DMA2 channel 5
HAL_DMA2_CH5_USART3_TX	Remap USART3 Tx on DMA2 channel 5
HAL_DMA2_CH5_USART4_TX	Remap USART4 Tx on DMA2 channel 5
HAL_DMA2_CH5_USART5_TX	Remap USART5 Tx on DMA2 channel 5
HAL_DMA2_CH5_USART6_TX	Remap USART6 Tx on DMA2 channel 5
HAL_DMA2_CH5_USART7_TX	Remap USART7 Tx on DMA2 channel 5
HAL_DMA2_CH5_USART8_TX	Remap USART8 Tx on DMA2 channel 5

IS\_HAL\_DMA1\_REMAP

IS\_HAL\_DMA2\_REMAP

**DMAEx Exported Macros**

- \_\_HAL\_DMA\_GET\_TC\_FLAG\_INDEX **Description:**
- Returns the current DMA Channel transfer complete flag.

**Parameters:**

- \_\_HANDLE\_\_: DMA handle

**Return value:**

- The: specified transfer complete 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\_TCIFx: Transfer complete flag
  - DMA\_FLAG\_HTIFx: Half transfer complete flag
  - DMA\_FLAG\_TEIFx: Transfer error flag Where x can be 0\_4, 1\_5, 2\_6 or 3\_7 to select the DMA Channel flag.

**Return value:**

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

`__HAL_DMA_CLEAR_FLAG`

**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_TCIFx`: Transfer complete flag
  - `DMA_FLAG_HTIFx`: Half transfer complete flag
  - `DMA_FLAG_TEIFx`: Transfer error flag Where x can be 0\_4, 1\_5, 2\_6 or 3\_7 to select the DMA Channel flag.

**Return value:**

- None:

`__HAL_DMA1_REMAP`

`__HAL_DMA2_REMAP`

## 16 HAL FLASH Generic Driver

### 16.1 FLASH Firmware driver registers structures

#### 16.1.1 FLASH\_EraseInitTypeDef

*FLASH\_EraseInitTypeDef* is defined in the `stm32f0xx_hal_flash.h`

##### Data Fields

- *uint32\_t TypeErase*
- *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 [\*FLASH\\_Type\\_Erase\*](#)
- *uint32\_t FLASH\_EraseInitTypeDef::PageAddress* PageAddress: Initial FLASH page address to erase when mass erase is disabled This parameter must be a value of [\*FLASHEx\\_Address\*](#)
- *uint32\_t FLASH\_EraseInitTypeDef::NbPages* NbPages: Number of pages to be erased. This parameter must be a value between 1 and (max number of pages - value of initial page)

#### 16.1.2 FLASH\_OBProgramInitTypeDef

*FLASH\_OBProgramInitTypeDef* is defined in the `stm32f0xx_hal_flash.h`

##### Data Fields

- *uint32\_t OptionType*
- *uint32\_t WRPState*
- *uint32\_t WRPPage*
- *uint8\_t RDPLevel*
- *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 [\*FLASH\\_OB\\_Type\*](#)
- *uint32\_t FLASH\_OBProgramInitTypeDef::WRPState* WRPState: Write protection activation or deactivation. This parameter can be a value of [\*FLASH\\_OB\\_WRP\\_State\*](#)
- *uint32\_t FLASH\_OBProgramInitTypeDef::WRPPage* WRPSector: specifies the page(s) to be write protected This parameter can be a value of [\*FLASHEx\\_OB\\_Write\\_Protection\*](#)
- *uint8\_t FLASH\_OBProgramInitTypeDef::RDPLevel* RDPLevel: Set the read protection level.. This parameter can be a value of [\*FLASH\\_OB\\_Read\\_Protection\*](#)

- `uint8_t FLASH_OBProgramInitTypeDef::USERConfig` USERConfig: Program the FLASH User Option Byte: IWDG / STOP / STDBY / BOOT1 / VDDA\_ANALOG / SRAM\_PARITY This parameter can be a combination of `FLASH_OB_Watchdog`, `FLASH_OB_nRST_STOP`, `FLASH_OB_nRST_STDBY`, `FLASH_OB_BOOT1`, `FLASH_OB_VDDA_Analog_Monitoring` and `FLASH_OB_RAM_Parity_Check_Enable`
- `uint32_t FLASH_OBProgramInitTypeDef::DATAAddress` DATAAddress: Address of the option byte DATA to be programmed This parameter can be a value of `FLASH_OB_Data_Address`
- `uint8_t FLASH_OBProgramInitTypeDef::DATAData` DATAData: Data to be stored in the option byte DATA This parameter can have any value

### 16.1.3 `FLASH_ProcessTypeDef`

`FLASH_ProcessTypeDef` is defined in the `stm32f0xx_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 FLASH_ErrorTypeDef 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 FLASH_ErrorTypeDef 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 STM32F0xx 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 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 Errors functions

This subsection permit to get in run-time Errors 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 FLASH 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	<b>HAL_StatusTypeDef HAL_Status</b>
Notes	<ul style="list-style-type: none"> <li>• The function <b>HAL_FLASH_Unlock()</b> should be called before to unlock the FLASH interface The function <b>HAL_FLASH_Lock()</b> 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 programmation (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 FLASH Type Program</li> </ul>

- |               |                                                                                                                                                                                                                                                                                                                                                                     |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Return values | <ul style="list-style-type: none"> <li>• <b>Address</b> : Specifies the address to be programmed.</li> <li>• <b>Data</b> : Specifies the data to be programmed</li> <li>• <b>HAL_StatusTypeDef HAL_Status</b></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>
Notes	<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           <ul style="list-style-type: none"> <li>– <b>Mass Erase</b> No return value expected</li> <li>– <b>Pages Erase</b> Address of the page which has been erased</li> <li>– <b>Program</b> Address which was selected for data program</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>none</b></li> </ul>
Notes	<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<ul style="list-style-type: none"><li>– <b>Mass Erase</b> No return value expected</li><li>– <b>Pages Erase</b> Address of the page which returned an error</li><li>– <b>Program</b> Address which was selected for data program</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>none</b></li></ul>
Notes	<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>• <b>HAL Status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</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>• <b>HAL Status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</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>• <b>HAL Status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</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>• <b>HAL Status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 16.2.16 HAL\_FLASH\_GetError

Function Name	<b>FLASH_ErrorTypeDef HAL_FLASH_GetError ( void )</b>
Function Description	Get the specific FLASH error flag.

---

Return values	<ul style="list-style-type: none"> <li>• <b>FLASH_ErrorCode</b> : The returned value can be:           <ul style="list-style-type: none"> <li>– <b>FLASH_ERROR_PG</b>: <i>FLASH Programming error flag</i></li> <li>– <b>FLASH_ERROR_WRP</b>: <i>FLASH Write protected error flag</i></li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 16.3 FLASH Firmware driver defines

### 16.3.1 FLASH

FLASH

***FLASH Flag definition***

<code>FLASH_FLAG_BSY</code>	FLASH Busy flag
<code>FLASH_FLAG_PGERR</code>	FLASH Programming error flag
<code>FLASH_FLAG_WRPERR</code>	FLASH Write protected error flag
<code>FLASH_FLAG_EOP</code>	FLASH End of Operation flag

***FLASH Interrupt***

<code>_HAL_FLASH_ENABLE_IT</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• Enable the specified FLASH interrupt.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>_INTERRUPT_</code>: FLASH interrupt This parameter can be any combination of the following values:           <ul style="list-style-type: none"> <li>– <code>FLASH_IT_EOP</code>: End of FLASH Operation Interrupt</li> <li>– <code>FLASH_IT_ERR</code>: Error Interrupt</li> </ul> </li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>• none:</li> </ul>
<code>_HAL_FLASH_DISABLE_IT</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• Disable the specified FLASH interrupt.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>_INTERRUPT_</code>: FLASH interrupt This parameter can be any combination of the following values:           <ul style="list-style-type: none"> <li>– <code>FLASH_IT_EOP</code>: End of FLASH Operation Interrupt</li> <li>– <code>FLASH_IT_ERR</code>: Error Interrupt</li> </ul> </li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>• none:</li> </ul>
<code>_HAL_FLASH_GET_FLAG</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• Get the specified FLASH flag status.</li> </ul>

**Parameters:**

- \_\_FLAG\_\_: specifies the FLASH flag to check. This parameter can be one of the following values:
  - FLASH\_FLAG\_EOP : FLASH End of Operation flag
  - FLASH\_FLAG\_WRPERR: FLASH Write protected error flag
  - FLASH\_FLAG\_PGERR : FLASH Programming error flag
  - FLASH\_FLAG\_BSY : FLASH Busy flag

**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 : FLASH End of Operation flag
  - FLASH\_FLAG\_WRPERR: FLASH Write protected error flag
  - FLASH\_FLAG\_PGERR : FLASH Programming error flag

**Return value:**

- none:

***FLASH Interrupt definition***

**FLASH\_IT\_EOP**      End of FLASH Operation Interrupt source

**FLASH\_IT\_ERR**      Error Interrupt source

***FLASH Latency***

**FLASH\_LATENCY\_0**      FLASH Zero Latency cycle

**FLASH\_LATENCY\_1**      FLASH One Latency cycle

**IS\_FLASH\_LATENCY**

[\\_\\_HAL\\_FLASH\\_SET\\_LATENCY](#)    **Description:**

- Set the FLASH Latency.

**Parameters:**

- \_\_LATENCY\_\_: FLASH Latency The value of this parameter depend on device used within the same series

**Return value:**

- None:

***FLASH OB BOOT1***

OB_BOOT1_RESET	BOOT1 Reset
OB_BOOT1_SET	BOOT1 Set
IS_OB_BOOT1	
<b>FLASH OB Data Address</b>	
IS_OB_DATA_ADDRESS	
<b>FLASH OB nRST STDBY</b>	
OB_STDBY_NO_RST	No reset generated when entering in STANDBY
OB_STDBY_RST	Reset generated when entering in STANDBY
IS_OB_STDBY_SOURCE	
<b>FLASH OB nRST STOP</b>	
OB_STOP_NO_RST	No reset generated when entering in STOP
OB_STOP_RST	Reset generated when entering in STOP
IS_OB_STOP_SOURCE	
<b>FLASH OB RAM Parity Check Enable</b>	
OB_RAM_PARITY_CHECK_SET	RAM parity check enable set
OB_RAM_PARITY_CHECK_RESET	RAM parity check enable reset
IS_OB_SRAM_PARITY	
<b>FLASH OB Read Protection</b>	
OB_RDP_LEVEL_0	
OB_RDP_LEVEL_1	
OB_RDP_LEVEL_2	Warning: When enabling read protection level 2 it's no more possible to go back to level 1 or 0
IS_OB_RDP_LEVEL	
<b>FLASH Option Bytes Type</b>	
OPTIONBYTE_WRP	WRP option byte configuration
OPTIONBYTE_RDP	RDP option byte configuration
OPTIONBYTE_USER	USER option byte configuration
OPTIONBYTE_DATA	DATA option byte configuration
IS_OPTIONBYTE	
<b>FLASH OB VDDA Analog Monitoring</b>	
OB_VDDA_ANALOG_ON	Analog monitoring on VDDA Power source ON
OB_VDDA_ANALOG_OFF	Analog monitoring on VDDA Power source OFF
IS_OB_VDDA_ANALOG	
<b>FLASH OB Watchdog</b>	
OB_WDG_SW	Software WDG selected
OB_WDG_HW	Hardware WDG selected
IS_OB_WDG_SOURCE	

***FLASH WRP State***

WRPSTATE\_DISABLE Disable the write protection of the desired pages

WRPSTATE\_ENABLE Enable the write protection of the desired pages

IS\_WRPSTATE

***FLASH Prefetch***

`__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 Timeout definition***

`HAL_FLASH_TIMEOUT_VALUE`

***FLASH Type Erase***

`TYPEERASE_PAGES` Pages erase only

`TYPEERASE_MASSERASE` Flash mass erase activation

`IS_TYPEERASE`

***FLASH Type Program***

`TYPEPROGRAM_HALFWORD` Program a half-word (16-bit) at a specified address.

`TYPEPROGRAM_WORD` Program a word (32-bit) at a specified address.

`TYPEPROGRAM_DOUBLEWORD` Program a double word (64-bit) at a specified address

`IS_TYPEPROGRAM`

## 17 HAL FLASH Extension Driver

### 17.1 FLASHEx Firmware driver API description

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

#### 17.1.1 Flash peripheral extended features

#### 17.1.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32F0xxx devices. It includes

- Set/Reset the write protection
- Program the user Option Bytes
- Get the Read protection Level

#### 17.1.3 IO operation functions

- `HAL_FLASHEx_Erase()`
- `HAL_FLASHEx_Erase_IT()`

#### 17.1.4 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()`

#### 17.1.5 `HAL_FLASHEx_Erase`

Function Name	<code>HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)</code>
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 <code>FLASH_EraseInitTypeDef</code> 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><b>HAL_StatusTypeDef HAL_Status</b></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>

### 17.1.6 HAL\_FLASHEx\_Erase\_IT

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (  FLASH_EraselInitTypeDef * pEraselInit)</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>pEraselInit</b> : pointer to an FLASH_EraselInitTypeDef structure that contains the configuration information for the erasing.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL_StatusTypeDef HAL_Status</b></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>

### 17.1.7 HAL\_FLASHEx\_OBErase

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_OBErase ( void )</b>
Function Description	Erases the FLASH option bytes.
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></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>

### 17.1.8 HAL\_FLASHEx\_OBProgram

Function Name	<code>HAL_StatusTypeDef HAL_FLASHEx_OBProgram (</code> <code>FLASH_OBProgramInitTypeDef * pOBInit)</code>
Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none"> <li>• <code>pOBInit</code> : pointer to an <code>FLASH_OBInitStruct</code> structure that contains the configuration information for the programming.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <code>HAL_StatusTypeDef HAL_Status</code></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The function <code>HAL_FLASH_Unlock()</code> should be called before to unlock the FLASH interface The function <code>HAL_FLASH_OB_Unlock()</code> should be called before to unlock the options bytes The function <code>HAL_FLASH_OB_Launch()</code> should be called after to force the reload of the options bytes (system reset will occur)</li> </ul>

### 17.1.9 HAL\_FLASHEx\_OBGetConfig

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

## 17.2 FLASHEx Firmware driver defines

### 17.2.1 FLASHEx

FLASHEx

**FLASHEx Address**

`IS_FLASH_PROGRAM_ADDRESS`

**FLASHEx Nb Pages**



IS\_FLASH\_NB\_PAGES

**FLASHEx OB BOOT0**

OB\_BOOT0\_RESET    BOOT0 Reset

OB\_BOOT0\_SET      BOOT0 Set

IS\_OB\_BOOT0

**FLASHEx OB BOOT SEL**

OB\_BOOT\_SEL\_RESET    BOOT\_SEL Reset

OB\_BOOT\_SEL\_SET      BOOT\_SEL Set

IS\_OB\_BOOT\_SEL

**FLASHEx OB Write Protection**

OB\_WRP\_PAGES0TO1

OB\_WRP\_PAGES2TO3

OB\_WRP\_PAGES4TO5

OB\_WRP\_PAGES6TO7

OB\_WRP\_PAGES8TO9

OB\_WRP\_PAGES10TO11

OB\_WRP\_PAGES12TO13

OB\_WRP\_PAGES14TO15

OB\_WRP\_PAGES16TO17

OB\_WRP\_PAGES18TO19

OB\_WRP\_PAGES20TO21

OB\_WRP\_PAGES22TO23

OB\_WRP\_PAGES24TO25

OB\_WRP\_PAGES26TO27

OB\_WRP\_PAGES28TO29

OB\_WRP\_PAGES30TO31

OB\_WRP\_PAGES32TO33

OB\_WRP\_PAGES34TO35

OB\_WRP\_PAGES36TO37

OB\_WRP\_PAGES38TO39

OB\_WRP\_PAGES40TO41

OB\_WRP\_PAGES42TO43

OB\_WRP\_PAGES44TO45

OB\_WRP\_PAGES46TO47

OB\_WRP\_PAGES48TO49

OB\_WRP\_PAGES50TO51

OB_WRP_PAGES52TO53	
OB_WRP_PAGES54TO55	
OB_WRP_PAGES56TO57	
OB_WRP_PAGES58TO59	
OB_WRP_PAGES60TO61	
OB_WRP_PAGES62TO127	
OB_WRP_PAGES0TO15MASK	
OB_WRP_PAGES16TO31MASK	
OB_WRP_PAGES32TO47MASK	
OB_WRP_PAGES48TO127MASK	
OB_WRP_ALLPAGES	Write protection of all pages
IS_OB_WRP	
<b><i>FLASHEx Page Size</i></b>	
FLASH_PAGE_SIZE	
<b><i>FLASHEx Private Constants</i></b>	
HAL_FLASH_TIMEOUT_VALUE	

## 18 HAL GPIO Generic Driver

### 18.1 GPIO Firmware driver registers structures

#### 18.1.1 GPIO\_InitTypeDef

`GPIO_InitTypeDef` is defined in the `stm32f0xx_hal_gpio.h`

##### Data Fields

- `uint32_t Pin`
- `uint32_t Mode`
- `uint32_t Pull`
- `uint32_t Speed`
- `uint32_t Alternate`

##### Field Documentation

- `uint32_t GPIO_InitTypeDef::Pin` Specifies the GPIO pins to be configured. This parameter can be any value of [GPIO\\_pins](#)
- `uint32_t GPIO_InitTypeDef::Mode` Specifies the operating mode for the selected pins. This parameter can be a value of [GPIO\\_mode](#)
- `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](#)
- `uint32_t GPIO_InitTypeDef::Speed` Specifies the speed for the selected pins. This parameter can be a value of [GPIO\\_speed](#)
- `uint32_t GPIO_InitTypeDef::Alternate` Peripheral to be connected to the selected pins. This parameter can be a value of [GPIOEx\\_Alternate\\_function\\_selection](#)

### 18.2 GPIO Firmware driver API description

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

#### 18.2.1 GPIO Peripheral features

Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:

- Input mode
- Analog mode
- Output mode
- Alternate function mode
- External interrupt/event lines

During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.

All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.

In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.

The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral's alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.

All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.

The external interrupt/event controller consists of up to 28 edge detectors (depending on products 16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently).

## 18.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function : \_\_GPIOx\_CLK\_ENABLE().
2. Configure the GPIO pin(s) using HAL\_GPIO\_Init().
  - Configure the IO mode using "Mode" member from GPIO\_InitTypeDef structure
  - Activate Pull-up, Pull-down resistor using "Pull" member from GPIO\_InitTypeDef structure.
  - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from GPIO\_InitTypeDef structure, the speed is configurable: Low, Medium and High.
  - If alternate mode is selected, the alternate function connected to the IO is configured through "Alternate" member from GPIO\_InitTypeDef structure
  - Analog mode is required when a pin is to be used as ADC channel or DAC output.
  - In case of external interrupt/event selection the "Mode" member from GPIO\_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL\_NVIC\_SetPriority() and enable it using HAL\_NVIC\_EnableIRQ().
4. HAL\_GPIO\_Delnit allows to set register values to their reset value. It's also to use it to unconfigure pin which was used as an external interrupt or in event mode. That's the only way to reset corresponding bit in EXTI & SYSCFG registers.
5. To get the level of a pin configured in input mode use HAL\_GPIO\_ReadPin().
6. To set/reset the level of a pin configured in output mode use HAL\_GPIO\_WritePin()/HAL\_GPIO\_TogglePin().
7. To lock pin configuration until next reset use HAL\_GPIO\_LockPin().
8. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
9. 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.
10. 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 de-initialization functions

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

#### 18.2.4 IO operation functions

- `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	<code>void HAL_GPIO_Init ( GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_InitStruct)</code>
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the <code>GPIO_InitStruct</code> .
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b> : where x can be (A..F) to select the GPIO peripheral for STM32F0 family</li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIO_InitStruct</b> : pointer to a <code>GPIO_InitTypeDef</code> 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>
Notes	<ul style="list-style-type: none"> <li>• GPIOD is only available on STM32F05xx, STM32F07xx and STM32F09xx</li> <li>• GPIOE is only available on STM32F07xx and STM32F09xx</li> </ul>

#### 18.2.6 HAL\_GPIO\_DeInit

Function Name	<code>void HAL_GPIO_DeInit ( GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)</code>
Function Description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b> : where x can be (A..F) to select the GPIO peripheral for STM32F0 family</li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIO_Pin</b> : specifies the port bit to be written. This parameter can be one of <code>GPIO_PIN_x</code> where x can be (0..15).</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>GPIOD is only available on STM32F05xx, STM32F07xx and STM32F09xx</li> <li>GPIOE is only available on STM32F07xx and STM32F09xx</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..F) to select the GPIO peripheral for STM32F0 family</li> </ul>
Parameters	<ul style="list-style-type: none"> <li><b>GPIO_Pin</b> : specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>The input port pin value.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>GPIOD is only available on STM32F05xx, STM32F07xx and STM32F09xx</li> <li>GPIOE is only available on STM32F07xx and STM32F09xx</li> </ul>

### 18.2.8 HAL\_GPIO\_WritePin

Function Name	<b>void HAL_GPIO_WritePin ( GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)</b>
Function Description	Sets or clears the selected data port bit.
Parameters	<ul style="list-style-type: none"> <li><b>GPIOx</b> : where x can be (A..F) to select the GPIO peripheral for STM32F0 family</li> </ul>
Parameters	<ul style="list-style-type: none"> <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> <li><b>PinState</b> : specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: <ul style="list-style-type: none"> <li><b>GPIO_PIN_RESET</b> to clear the port pin</li> <li><b>GPIO_PIN_SET</b> to set the port pin</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>

**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.
- GPIOD is only available on STM32F05xx, STM32F07xx and STM32F09xx
- GPIOE is only available on STM32F07xx and STM32F09xx

**18.2.9 HAL\_GPIO\_TogglePin**

Function Name	<b>void HAL_GPIO_TogglePin ( GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</b>
Function Description	Toggles the specified GPIO pin.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b> : where x can be (A..F) to select the GPIO peripheral for STM32F0 family</li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIO_Pin</b> : specifies the pins to be toggled.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• GPIOD is only available on STM32F05xx, STM32F07xx and STM32F09xx</li> <li>• GPIOE is only available on STM32F07xx and STM32F09xx</li> </ul>

**18.2.10 HAL\_GPIO\_LockPin**

Function Name	<b>HAL_StatusTypeDef HAL_GPIO_LockPin ( GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</b>
Function Description	Locks GPIO Pins configuration registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b> : where x can be (A..F) to select the GPIO peripheral for STM32F0 family</li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIO_Pin</b> : specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.</li> <li>• The configuration of the locked GPIO pins can no longer be</li> </ul>

- modified until the next reset.
- GPIOD is only available on STM32F05xx, STM32F07xx and STM32F09xx
- GPIOE is only available on STM32F07xx and STM32F09xx

### 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 port pin connected to corresponding EXTI line.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<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 callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>GPIO_Pin</b> : Specifies the port pin connected to corresponding EXTI line.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 18.3 GPIO Firmware driver defines

### 18.3.1 GPIO

GPIO

**GPIO Exported Macros**

<b>_HAL_GPIO_EXTI_GET_FLAG</b>	<b>Description:</b>
--------------------------------	---------------------

- 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).

**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:

**GPIO mode**

<code>GPIO_MODE_INPUT</code>	Input Floating Mode
<code>GPIO_MODE_OUTPUT_PP</code>	Output Push Pull Mode
<code>GPIO_MODE_OUTPUT_OD</code>	Output Open Drain Mode
<code>GPIO_MODE_AF_PP</code>	Alternate Function Push Pull Mode
<code>GPIO_MODE_AF_OD</code>	Alternate Function Open Drain Mode
<code>GPIO_MODE_ANALOG</code>	Analog Mode
<code>GPIO_MODE_IT_RISING</code>	External Interrupt Mode with Rising edge trigger detection
<code>GPIO_MODE_IT_FALLING</code>	External Interrupt Mode with Falling edge trigger detection
<code>GPIO_MODE_IT_RISING_FALLING</code>	External Interrupt Mode with Rising/Falling edge trigger detection
<code>GPIO_MODE_EVT_RISING</code>	External Event Mode with Rising edge trigger detection
<code>GPIO_MODE_EVT_FALLING</code>	External Event Mode with Falling edge trigger detection
<code>GPIO_MODE_EVT_RISING_FALLING</code>	External Event Mode with Rising/Falling edge trigger detection

`IS_GPIO_MODE`

**GPIO pins**

<code>GPIO_PIN_0</code>
<code>GPIO_PIN_1</code>
<code>GPIO_PIN_2</code>
<code>GPIO_PIN_3</code>
<code>GPIO_PIN_4</code>
<code>GPIO_PIN_5</code>
<code>GPIO_PIN_6</code>
<code>GPIO_PIN_7</code>
<code>GPIO_PIN_8</code>
<code>GPIO_PIN_9</code>
<code>GPIO_PIN_10</code>
<code>GPIO_PIN_11</code>
<code>GPIO_PIN_12</code>

GPIO\_PIN\_13  
GPIO\_PIN\_14  
GPIO\_PIN\_15  
GPIO\_PIN\_All  
GPIO\_PIN\_MASK  
IS\_GPIO\_PIN

***GPIO pin actions***

IS\_GPIO\_PIN\_ACTION

***GPIO Private Define***

GPIO\_MODE  
EXTI\_MODE  
GPIO\_MODE\_IT  
GPIO\_MODE\_EVT  
RISING\_EDGE  
FALLING\_EDGE  
GPIO\_OUTPUT\_TYPE  
GPIO\_NUMBER

***GPIO pull***

GPIO_NOPULL	No Pull-up or Pull-down activation
GPIO_PULLUP	Pull-up activation
GPIO_PULLDOWN	Pull-down activation

IS\_GPIO\_PULL

***GPIO speed***

GPIO_SPEED_LOW	Low speed
GPIO_SPEED_MEDIUM	Medium speed
GPIO_SPEED_HIGH	High speed

IS\_GPIO\_SPEED

## 19 HAL GPIO Extension Driver

### 19.1 GPIOEx Firmware driver defines

#### 19.1.1 GPIOEx

GPIOEx

***GPIOEx Alternate function selection***

GPIO_AF0_EVENTOUT	AF0: EVENTOUT Alternate Function mapping
GPIO_AF0_SWDIO	AF0: SWDIO Alternate Function mapping
GPIO_AF0_SWCLK	AF0: SWCLK Alternate Function mapping
GPIO_AF0_MCO	AF0: MCO Alternate Function mapping
GPIO_AF0_CEC	AF0: CEC Alternate Function mapping
GPIO_AF0_CRS	AF0: CRS Alternate Function mapping
GPIO_AF0_IR	AF0: IR Alternate Function mapping
GPIO_AF0_SPI1	AF0: SPI1/I2S1 Alternate Function mapping
GPIO_AF0_SPI2	AF0: SPI2/I2S2 Alternate Function mapping
GPIO_AF0_TIM1	AF0: TIM1 Alternate Function mapping
GPIO_AF0_TIM3	AF0: TIM3 Alternate Function mapping
GPIO_AF0_TIM14	AF0: TIM14 Alternate Function mapping
GPIO_AF0_TIM15	AF0: TIM15 Alternate Function mapping
GPIO_AF0_TIM16	AF0: TIM16 Alternate Function mapping
GPIO_AF0_TIM17	AF0: TIM17 Alternate Function mapping
GPIO_AF0_TSC	AF0: TSC Alternate Function mapping
GPIO_AF0_USART1	AF0: USART1 Alternate Function mapping
GPIO_AF0_USART2	AF0: USART2 Alternate Function mapping
GPIO_AF0_USART3	AF0: USART3 Alternate Function mapping
GPIO_AF0_USART4	AF0: USART4 Alternate Function mapping
GPIO_AF0_USART8	AF0: USART8 Alternate Function mapping
GPIO_AF0_CAN	AF0: CAN Alternate Function mapping
GPIO_AF1_TIM3	AF1: TIM3 Alternate Function mapping
GPIO_AF1_TIM15	AF1: TIM15 Alternate Function mapping
GPIO_AF1_USART1	AF1: USART1 Alternate Function mapping
GPIO_AF1_USART2	AF1: USART2 Alternate Function mapping
GPIO_AF1_USART3	AF1: USART3 Alternate Function mapping
GPIO_AF1_USART4	AF1: USART4 Alternate Function mapping
GPIO_AF1_USART5	AF1: USART5 Alternate Function mapping

GPIO_AF1_USART6	AF1: USART6 Alternate Function mapping
GPIO_AF1_USART7	AF1: USART7 Alternate Function mapping
GPIO_AF1_USART8	AF1: USART8 Alternate Function mapping
GPIO_AF1_IR	AF1: IR Alternate Function mapping
GPIO_AF1_CEC	AF1: CEC Alternate Function mapping
GPIO_AF1_EVENTOUT	AF1: EVENTOUT Alternate Function mapping
GPIO_AF1_I2C1	AF1: I2C1 Alternate Function mapping
GPIO_AF1_I2C2	AF1: I2C2 Alternate Function mapping
GPIO_AF1_TSC	AF1: TSC Alternate Function mapping
GPIO_AF1_SPI1	AF1: SPI1 Alternate Function mapping
GPIO_AF1_SPI2	AF1: SPI2 Alternate Function mapping
GPIO_AF2_TIM1	AF2: TIM1 Alternate Function mapping
GPIO_AF2_TIM2	AF2: TIM2 Alternate Function mapping
GPIO_AF2_TIM16	AF2: TIM16 Alternate Function mapping
GPIO_AF2_TIM17	AF2: TIM17 Alternate Function mapping
GPIO_AF2_EVENTOUT	AF2: EVENTOUT Alternate Function mapping
GPIO_AF2_USART5	AF2: USART5 Alternate Function mapping
GPIO_AF2_USART6	AF2: USART6 Alternate Function mapping
GPIO_AF2_USART7	AF2: USART7 Alternate Function mapping
GPIO_AF2_USART8	AF2: USART8 Alternate Function mapping
GPIO_AF3_EVENTOUT	AF3: EVENTOUT Alternate Function mapping
GPIO_AF3_TSC	AF3: TSC Alternate Function mapping
GPIO_AF3_TIM15	AF3: TIM15 Alternate Function mapping
GPIO_AF3_I2C1	AF3: I2C1 Alternate Function mapping
GPIO_AF4_TIM14	AF4: TIM14 Alternate Function mapping
GPIO_AF4_USART4	AF4: USART4 Alternate Function mapping
GPIO_AF4_USART3	AF4: USART3 Alternate Function mapping
GPIO_AF4_CRS	AF4: CRS Alternate Function mapping
GPIO_AF4_CAN	AF4: CAN Alternate Function mapping
GPIO_AF4_I2C1	AF4: I2C1 Alternate Function mapping
GPIO_AF4_USART5	AF4: USART5 Alternate Function mapping
GPIO_AF5_TIM15	AF5: TIM15 Alternate Function mapping
GPIO_AF5_TIM16	AF5: TIM16 Alternate Function mapping
GPIO_AF5_TIM17	AF5: TIM17 Alternate Function mapping
GPIO_AF5_SPI2	AF5: SPI2 Alternate Function mapping
GPIO_AF5_I2C2	AF5: I2C2 Alternate Function mapping

GPIO_AF5_MCO	AF5: MCO Alternate Function mapping
GPIO_AF5_USART6	AF5: USART6 Alternate Function mapping
GPIO_AF6_EVENTOUT	AF6: EVENTOUT Alternate Function mapping
GPIO_AF7_COMP1	AF7: COMP1 Alternate Function mapping
GPIO_AF7_COMP2	AF7: COMP2 Alternate Function mapping
IS_GPIO_AF	
<b><i>GPIOEx_Get Port Index</i></b>	
GET_GPIO_INDEX	

## 20 HAL I2C Generic Driver

### 20.1 I2C Firmware driver registers structures

#### 20.1.1 I2C\_InitTypeDef

*I2C\_InitTypeDef* is defined in the `stm32f0xx_hal_i2c.h`

##### Data Fields

- *uint32\_t Timing*
- *uint32\_t OwnAddress1*
- *uint32\_t AddressingMode*
- *uint32\_t DualAddressMode*
- *uint32\_t OwnAddress2*
- *uint32\_t OwnAddress2Masks*
- *uint32\_t GeneralCallMode*
- *uint32\_t NoStretchMode*

##### Field Documentation

- ***uint32\_t I2C\_InitTypeDef::Timing*** Specifies the I2C\_TIMINGR\_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- ***uint32\_t I2C\_InitTypeDef::OwnAddress1*** Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32\_t I2C\_InitTypeDef::AddressingMode*** Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of [\*I2C\\_addressing\\_mode\*](#)
- ***uint32\_t I2C\_InitTypeDef::DualAddressMode*** Specifies if dual addressing mode is selected. This parameter can be a value of [\*I2C\\_dual\\_addressing\\_mode\*](#)
- ***uint32\_t I2C\_InitTypeDef::OwnAddress2*** Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32\_t I2C\_InitTypeDef::OwnAddress2Masks*** Specifies the acknowledge mask address second device own address if dual addressing mode is selected This parameter can be a value of [\*I2C\\_own\\_address2\\_masks\*](#).
- ***uint32\_t I2C\_InitTypeDef::GeneralCallMode*** Specifies if general call mode is selected. This parameter can be a value of [\*I2C\\_general\\_call\\_addressing\\_mode\*](#).
- ***uint32\_t I2C\_InitTypeDef::NoStretchMode*** Specifies if nostretch mode is selected. This parameter can be a value of [\*I2C\\_nostretch\\_mode\*](#)

#### 20.1.2 I2C\_HandleTypeDef

*I2C\_HandleTypeDef* is defined in the `stm32f0xx_hal_i2c.h`

##### Data Fields

- *I2C\_TypeDef \* Instance*
- *I2C\_InitTypeDef Init*
- *uint8\_t \* pBuffPtr*
- *uint16\_t XferSize*
- *\_\_IO uint16\_t XferCount*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*

- ***HAL\_LockTypeDef Lock***
- ***\_IO HAL\_I2C\_StateTypeDef State***
- ***\_IO HAL\_I2C\_ErrorTypeDef ErrorCode***

#### Field Documentation

- ***I2C\_TypeDef\* I2C\_HandleTypeDef::Instance*** I2C registers base address
- ***I2C\_InitTypeDef I2C\_HandleTypeDef::Init*** I2C communication parameters
- ***uint8\_t\* I2C\_HandleTypeDef::pBuffPtr*** Pointer to I2C transfer buffer
- ***uint16\_t I2C\_HandleTypeDef::XferSize*** I2C transfer size
- ***\_IO uint16\_t I2C\_HandleTypeDef::XferCount*** I2C transfer counter
- ***DMA\_HandleTypeDef\* I2C\_HandleTypeDef::hdmatx*** I2C Tx DMA handle parameters
- ***DMA\_HandleTypeDef\* I2C\_HandleTypeDef::hdmarx*** I2C Rx DMA handle parameters
- ***HAL\_LockTypeDef I2C\_HandleTypeDef::Lock*** I2C locking object
- ***\_IO HAL\_I2C\_StateTypeDef I2C\_HandleTypeDef::State*** I2C communication state
- ***\_IO HAL\_I2C\_ErrorTypeDef I2C\_HandleTypeDef::ErrorCode***

## 20.2 I2C Firmware driver API description

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

### 20.2.1 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:
  - Enable the I2Cx interface clock
  - I2C pins configuration
    - Enable the clock for the I2C GPIOs
    - Configure I2C pins as alternate function open-drain
  - NVIC configuration if you need to use interrupt process
    - Configure the I2Cx interrupt priority
    - Enable the NVIC I2C IRQ Channel
  - 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 Clock Timing, Own Address1, Master Addressing Mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
4. Initialize the I2C registers by calling the HAL\_I2C\_Init() API:

- These API's configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL\_I2C\_MspInit(&hi2c) API.
- 5. To check if target device is ready for communication, use the function HAL\_I2C\_IsDeviceReady()
- 6. For I2C IO and IO MEM operations, three modes of operations are available within this driver :

### Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL\_I2C\_Master\_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL\_I2C\_Master\_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL\_I2C\_Slave\_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL\_I2C\_Slave\_Receive()

### Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL\_I2C\_Mem\_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL\_I2C\_Mem\_Read()

### Interrupt mode IO operation

- Transmit in master mode an amount of data in non blocking mode using HAL\_I2C\_Master\_Transmit\_IT()
- At transmission end of transfer HAL\_I2C\_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode using HAL\_I2C\_Master\_Receive\_IT()
- At reception end of transfer HAL\_I2C\_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode using HAL\_I2C\_Slave\_Transmit\_IT()
- At transmission end of transfer HAL\_I2C\_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode using HAL\_I2C\_Slave\_Receive\_IT()
- At reception end of transfer HAL\_I2C\_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### Interrupt mode IO MEM operation

- Write an amount of data in no-blocking mode with Interrupt to a specific memory address using HAL\_I2C\_Mem\_Write\_IT()
- At MEM end of write transfer HAL\_I2C\_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemTxCpltCallback
- Read an amount of data in no-blocking mode with Interrupt from a specific memory address using HAL\_I2C\_Mem\_Read\_IT()
- At MEM end of read transfer HAL\_I2C\_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### DMA mode IO operation

- Transmit in master mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Master\_Transmit\_DMA()
- At transmission end of transfer HAL\_I2C\_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Master\_Receive\_DMA()
- At reception end of transfer HAL\_I2C\_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Slave\_Transmit\_DMA()
- At transmission end of transfer HAL\_I2C\_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Slave\_Receive\_DMA()
- At reception end of transfer HAL\_I2C\_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### DMA mode IO MEM operation

- Write an amount of data in no-blocking mode with DMA to a specific memory address using HAL\_I2C\_Mem\_Write\_DMA()
- At MEM end of write transfer HAL\_I2C\_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemTxCpltCallback
- Read an amount of data in no-blocking mode with DMA from a specific memory address using HAL\_I2C\_Mem\_Read\_DMA()
- At MEM end of read transfer HAL\_I2C\_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### 12C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- `_HAL_I2C_ENABLE`: Enable the I2C peripheral
- `_HAL_I2C_DISABLE`: Disable the I2C peripheral
- `_HAL_I2C_GET_FLAG` : Checks whether the specified I2C flag is set or not
- `_HAL_I2C_CLEAR_FLAG` : Clears the specified I2C pending flag
- `_HAL_I2C_ENABLE_IT`: Enables the specified I2C interrupt
- `_HAL_I2C_DISABLE_IT`: Disables the specified I2C interrupt



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

### 20.2.2 Initialization and Configuration functions

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

- User must Implement `HAL_I2C_MspInit()` function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function `HAL_I2C_Init()` to configure the selected device with the selected configuration:
  - Clock Timing
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
- Call the function `HAL_I2C_DeInit()` to restore the default configuration of the selected I2Cx peripheral.
- `HAL_I2C_Init()`
- `HAL_I2C_DeInit()`
- `HAL_I2C_MspInit()`
- `HAL_I2C_MspDeInit()`

### 20.2.3 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.
  - Non-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :

- HAL\_I2C\_Master\_Transmit()
  - HAL\_I2C\_Master\_Receive()
  - HAL\_I2C\_Slave\_Transmit()
  - HAL\_I2C\_Slave\_Receive()
  - HAL\_I2C\_Mem\_Write()
  - HAL\_I2C\_Mem\_Read()
  - HAL\_I2C\_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
    - HAL\_I2C\_Master\_Transmit\_IT()
    - HAL\_I2C\_Master\_Receive\_IT()
    - HAL\_I2C\_Slave\_Transmit\_IT()
    - HAL\_I2C\_Slave\_Receive\_IT()
    - HAL\_I2C\_Mem\_Write\_IT()
    - HAL\_I2C\_Mem\_Read\_IT()
  4. No-Blocking mode functions with DMA are :
    - HAL\_I2C\_Master\_Transmit\_DMA()
    - HAL\_I2C\_Master\_Receive\_DMA()
    - HAL\_I2C\_Slave\_Transmit\_DMA()
    - HAL\_I2C\_Slave\_Receive\_DMA()
    - HAL\_I2C\_Mem\_Write\_DMA()
    - HAL\_I2C\_Mem\_Read\_DMA()
  5. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
    - HAL\_I2C\_MemTxCpltCallback()
    - HAL\_I2C\_MemRxCpltCallback()
    - HAL\_I2C\_MasterTxCpltCallback()
    - HAL\_I2C\_MasterRxCpltCallback()
    - HAL\_I2C\_SlaveTxCpltCallback()
    - HAL\_I2C\_SlaveRxCpltCallback()
    - HAL\_I2C\_ErrorCallback()
    - *HAL\_I2C\_Master\_Transmit()*
    - *HAL\_I2C\_Master\_Receive()*
    - *HAL\_I2C\_Slave\_Transmit()*
    - *HAL\_I2C\_Slave\_Receive()*
    - *HAL\_I2C\_Master\_Transmit\_IT()*
    - *HAL\_I2C\_Master\_Receive\_IT()*
    - *HAL\_I2C\_Slave\_Transmit\_IT()*
    - *HAL\_I2C\_Slave\_Receive\_IT()*
    - *HAL\_I2C\_Master\_Transmit\_DMA()*
    - *HAL\_I2C\_Master\_Receive\_DMA()*
    - *HAL\_I2C\_Slave\_Transmit\_DMA()*
    - *HAL\_I2C\_Slave\_Receive\_DMA()*
    - *HAL\_I2C\_Mem\_Write()*
    - *HAL\_I2C\_Mem\_Read()*
    - *HAL\_I2C\_Mem\_Write\_IT()*
    - *HAL\_I2C\_Mem\_Read\_IT()*
    - *HAL\_I2C\_Mem\_Write\_DMA()*
    - *HAL\_I2C\_Mem\_Read\_DMA()*
    - *HAL\_I2C\_IsDeviceReady()*
    - *HAL\_I2C\_EV\_IRQHandler()*
    - *HAL\_I2C\_ER\_IRQHandler()*
    - *HAL\_I2C\_MasterTxCpltCallback()*
    - *HAL\_I2C\_MasterRxCpltCallback()*
    - *HAL\_I2C\_SlaveTxCpltCallback()*

- `HAL_I2C_SlaveRxCpltCallback()`
- `HAL_I2C_MemTxCpltCallback()`
- `HAL_I2C_MemRxCpltCallback()`
- `HAL_I2C_ErrorCallback()`

#### 20.2.4 Peripheral State and Errors functions

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

- `HAL_I2C_GetState()`
- `HAL_I2C_GetError()`

#### 20.2.5 HAL\_I2C\_Init

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Init ( I2C_HandleTypeDef * hi2c)</code>
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>• <code>hi2c</code> : 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 20.2.6 HAL\_I2C\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_I2C_DeInit ( I2C_HandleTypeDef * hi2c)</code>
Function Description	DeInitializes the I2C peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <code>hi2c</code> : 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 20.2.7 HAL\_I2C\_MspInit

Function Name	<b>void HAL_I2C_MspInit ( <i>I2C_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 20.2.8 HAL\_I2C\_MspDeInit

Function Name	<b>void HAL_I2C_MspDeInit ( <i>I2C_HandleTypeDef</i> * 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 contains the configuration information for the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 20.2.9 HAL\_I2C\_Master\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit ( <i>I2C_HandleTypeDef</i> * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmits in master 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>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> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.10 HAL\_I2C\_Master\_Receive

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Master_Receive ( I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)</code>
Function Description	Receives in master 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>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><li><b>Timeout</b> : Timeout duration</li></ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 20.2.11 HAL\_I2C\_Slave\_Transmit

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Slave_Transmit ( I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)</code>
Function Description	Transmits 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	<b>HAL status</b>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 20.2.12 HAL\_I2C\_Slave\_Receive

---

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive (</b> <b>I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size,</b> <b>uint32_t Timeout)</b>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.13 HAL\_I2C\_Master\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (</b> <b>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</b>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.14 HAL\_I2C\_Master\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (</b> <b>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</b>
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</li> </ul>

---

	contains the configuration information for the specified I2C.
• <b>DevAddress</b> : Target device address	
• <b>pData</b> : Pointer to data buffer	
• <b>Size</b> : Amount of data to be sent	
Return values	• <b>HAL status</b>
Notes	• None.

## 20.2.15 HAL\_I2C\_Slave\_Transmit\_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT ( I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</code>
Function Description	Transmit in slave mode an amount of data in no-blocking mode with 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	• <b>HAL status</b>
Notes	• None.

## 20.2.16 HAL\_I2C\_Slave\_Receive\_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT ( I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</code>
Function Description	Receive in slave mode an amount of data in no-blocking mode with 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	• <b>HAL status</b>
Notes	• None.

### 20.2.17 HAL\_I2C\_Master\_Transmit\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA ( I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</code>
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	• <b>HAL status</b>
Notes	• None.

### 20.2.18 HAL\_I2C\_Master\_Receive\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA ( I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</code>
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 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	• <b>HAL status</b>
Notes	• None.

### 20.2.19 HAL\_I2C\_Slave\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (</b> <b>I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</b>
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><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 20.2.20 HAL\_I2C\_Slave\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (</b> <b>I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</b>
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><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 20.2.21 HAL\_I2C\_Mem\_Write

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Mem_Write (</b> <b>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
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> </ul>

- **MemAddress** : Internal memory address
  - **MemAddSize** : Size of internal memory address
  - **pData** : Pointer to data buffer
  - **Size** : Amount of data to be sent
  - **Timeout** : Timeout duration
- Return values
- **HAL status**
- Notes
- None.

## 20.2.22 HAL\_I2C\_Mem\_Read

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size, uint32_t Timeout)</code>
Function Description	Read an amount of data in blocking mode from a specific memory address.
Parameters	<ul style="list-style-type: none"> <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> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 20.2.23 HAL\_I2C\_Mem\_Write\_IT

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

## 20.2.24 HAL\_I2C\_Mem\_Read\_IT

Function Name	<code>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)</code>
Function Description	Read an amount of data in no-blocking mode with Interrupt from a specific memory address.
Parameters	<ul style="list-style-type: none"> <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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 20.2.25 HAL\_I2C\_Mem\_Write\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA ( I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)</code>
Function Description	Write an amount of data in no-blocking mode with DMA to a specific memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : : Pointer to a I2C_HandleTypeDef structure that</li> </ul>

	contains the configuration information for the specified I2C.
• <b>DevAddress</b> : Target device address	
• <b>MemAddress</b> : Internal memory address	
• <b>MemAddSize</b> : Size of internal memory address	
• <b>pData</b> : Pointer to data buffer	
• <b>Size</b> : Amount of data to be sent	
Return values	• <b>HAL status</b>
Notes	• None.

## 20.2.26 HAL\_I2C\_Mem\_Read\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size)</code>
Function Description	Reads an amount of data in no-blocking mode with DMA from a specific memory address.
Parameters	<ul style="list-style-type: none"> <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	• <b>HAL status</b>
Notes	• None.

## 20.2.27 HAL\_I2C\_IsDeviceReady

Function Name	<code>HAL_StatusTypeDef HAL_I2C_IsDeviceReady (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t</code> <code>Trials, uint32_t Timeout)</code>
Function Description	Checks if target device is ready for communication.
Parameters	<ul style="list-style-type: none"> <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> </ul>

- |               |                                                                                                                                |
|---------------|--------------------------------------------------------------------------------------------------------------------------------|
| Return values | <ul style="list-style-type: none"><li>• <b>Trials</b> : Number of trials</li><li>• <b>Timeout</b> : Timeout duration</li></ul> |
| Notes         | <ul style="list-style-type: none"><li>• <b>HAL status</b></li><li>• This function is used with Memory devices</li></ul>        |

### 20.2.28 HAL\_I2C\_EV\_IRQHandler

Function Name	<b>void HAL_I2C_EV_IRQHandler ( <i>I2C_HandleTypeDef</i> * 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 <i>I2C_HandleTypeDef</i> structure that contains the configuration information for the specified I2C.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 20.2.29 HAL\_I2C\_ER\_IRQHandler

Function Name	<b>void HAL_I2C_ER_IRQHandler ( <i>I2C_HandleTypeDef</i> * 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 <i>I2C_HandleTypeDef</i> structure that contains the configuration information for the specified I2C.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 20.2.30 HAL\_I2C\_MasterTxCpltCallback

Function Name	<b>void HAL_I2C_MasterTxCpltCallback ( <i>I2C_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 20.2.31 HAL\_I2C\_MasterRxCpltCallback

Function Name	<b>void HAL_I2C_MasterRxCpltCallback ( <i>I2C_HandleTypeDef</i> * <b>hi2c</b>)</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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 20.2.32 HAL\_I2C\_SlaveTxCpltCallback

Function Name	<b>void HAL_I2C_SlaveTxCpltCallback ( <i>I2C_HandleTypeDef</i> * <b>hi2c</b>)</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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 20.2.33 HAL\_I2C\_SlaveRxCpltCallback

Function Name	<b>void HAL_I2C_SlaveRxCpltCallback ( <i>I2C_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 20.2.34 HAL\_I2C\_MemTxCpltCallback

Function Name	<b>void HAL_I2C_MemTxCpltCallback ( <i>I2C_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 20.2.35 HAL\_I2C\_MemRxCpltCallback

Function Name	<b>void HAL_I2C_MemRxCpltCallback ( <i>I2C_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 20.2.36 HAL\_I2C\_ErrorCallback

Function Name	<b>void HAL_I2C_ErrorCallback ( <i>I2C_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 20.2.37 HAL\_I2C\_GetState

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

### 20.2.38 HAL\_I2C\_GetError

Function Name	<b>uint32_t HAL_I2C_GetError ( <i>I2C_HandleTypeDef</i> * 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>• <b>I2C Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 20.3 I2C Firmware driver defines

### 20.3.1 I2C

I2C

***I2C addressing mode***

I2C\_ADDRESSINGMODE\_7BIT  
I2C\_ADDRESSINGMODE\_10BIT  
IS\_I2C\_ADDRESSING\_MODE

***I2C dual addressing mode***

I2C\_DUALADDRESS\_DISABLED  
I2C\_DUALADDRESS\_ENABLED  
IS\_I2C\_DUAL\_ADDRESS

***I2C Exported Macros***

<code>_HAL_I2C_RESET_HANDLE_STATE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Reset I2C handle state.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>_HANDLE_</code>: I2C handle.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None:</li></ul>
<code>_HAL_I2C_ENABLE_IT</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Enables or disables the specified I2C interrupts.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>_HANDLE_</code>: specifies the I2C Handle. This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.</li><li><code>_INTERRUPT_</code>: specifies the interrupt source to enable or disable. This parameter can be one of the following values:<ul style="list-style-type: none"><li><code>I2C_IT_ERRI</code>: Errors interrupt enable</li><li><code>I2C_IT_TCI</code>: Transfer complete interrupt enable</li><li><code>I2C_IT_STOPI</code>: STOP detection interrupt enable</li><li><code>I2C_IT_NACKI</code>: NACK received interrupt enable</li><li><code>I2C_IT_ADDRI</code>: Address match interrupt enable</li><li><code>I2C_IT_RXI</code>: RX interrupt enable</li><li><code>I2C_IT_TXI</code>: TX interrupt enable</li></ul></li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None:</li></ul>

`__HAL_I2C_DISABLE_IT`  
`__HAL_I2C_GET_IT_SOURCE`

**Description:**

- Checks if the specified I2C interrupt source is enabled or disabled.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle. This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.
- `__INTERRUPT__`: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - `I2C_IT_ERRI`: Errors interrupt enable
  - `I2C_IT_TCI`: Transfer complete interrupt enable
  - `I2C_IT_STOPI`: STOP detection interrupt enable
  - `I2C_IT_NACKI`: NACK received interrupt enable
  - `I2C_IT_ADDRI`: Address match interrupt enable
  - `I2C_IT_RXI`: RX interrupt enable
  - `I2C_IT_TXI`: TX interrupt enable

**Return value:**

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

`__HAL_I2C_GET_FLAG`

**Description:**

- Checks whether the specified I2C flag is set or not.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle. This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `I2C_FLAG_TXE`: Transmit data register empty
  - `I2C_FLAG_TXIS`: Transmit interrupt status
  - `I2C_FLAG_RXNE`: Receive data register not empty
  - `I2C_FLAG_ADDR`: Address matched (slave mode)
  - `I2C_FLAG_AF`: Acknowledge failure received flag
  - `I2C_FLAG_STOPF`: STOP detection flag
  - `I2C_FLAG_TC`: Transfer complete (master mode)
  - `I2C_FLAG_TCR`: Transfer complete

- reload
- I2C\_FLAG\_BERR: Bus error
  - I2C\_FLAG\_ARLO: Arbitration lost
  - I2C\_FLAG\_OVR: Overrun/Underrun
  - I2C\_FLAG\_PECERR: PEC error in reception
  - I2C\_FLAG\_TIMEOUT: Timeout or Tlow detection flag
  - I2C\_FLAG\_ALERT: SMBus alert
  - I2C\_FLAG\_BUSY: Bus busy
  - I2C\_FLAG\_DIR: Transfer direction (slave mode)

**Return value:**

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

**\_HAL\_I2C\_CLEAR\_FLAG****Description:**

- Clears the I2C pending flags which are cleared by writing 1 in a specific bit.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle. This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.
- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C\_FLAG\_ADDR: Address matched (slave mode)
  - I2C\_FLAG\_AF: Acknowledge failure flag
  - I2C\_FLAG\_STOPF: STOP detection flag
  - I2C\_FLAG\_BERR: Bus error
  - I2C\_FLAG\_ARLO: Arbitration lost
  - I2C\_FLAG\_OVR: Overrun/Underrun
  - I2C\_FLAG\_PECERR: PEC error in reception
  - I2C\_FLAG\_TIMEOUT: Timeout or Tlow detection flag
  - I2C\_FLAG\_ALERT: SMBus alert

**Return value:**

- None:

**\_HAL\_I2C\_ENABLE**  
**\_HAL\_I2C\_DISABLE**  
**\_HAL\_I2C\_RESET\_CR2**  
**\_HAL\_I2C\_MEM\_ADD\_MSB**  
**\_HAL\_I2C\_MEM\_ADD\_LSB**

`__HAL_I2C_GENERATE_START`

`IS_I2C_OWN_ADDRESS1`

`IS_I2C_OWN_ADDRESS2`

***I2C Flag definition***

`I2C_FLAG_TXE`

`I2C_FLAG_TXIS`

`I2C_FLAG_RXNE`

`I2C_FLAG_ADDR`

`I2C_FLAG_AF`

`I2C_FLAG_STOPF`

`I2C_FLAG_TC`

`I2C_FLAG_TCR`

`I2C_FLAG_BERR`

`I2C_FLAG_ARLO`

`I2C_FLAG_OVR`

`I2C_FLAG_PECERR`

`I2C_FLAG_TIMEOUT`

`I2C_FLAG_ALERT`

`I2C_FLAG_BUSY`

`I2C_FLAG_DIR`

***I2C general call addressing mode***

`I2C_GENERALCALL_DISABLED`

`I2C_GENERALCALL_ENABLED`

`IS_I2C_GENERAL_CALL`

***I2C Interrupt configuration definition***

`I2C_IT_ERRI`

`I2C_IT_TCI`

`I2C_IT_STOPI`

`I2C_IT_NACKI`

`I2C_IT_ADDRI`

`I2C_IT_RXI`

`I2C_IT_TXI`

***I2C Memory Address Size***

`I2C_MEMADD_SIZE_8BIT`

`I2C_MEMADD_SIZE_16BIT`

`IS_I2C_MEMADD_SIZE`

***I2C nostretch mode***

I2C\_NOSTRETCH\_DISABLED

I2C\_NOSTRETCH\_ENABLED

IS\_I2C\_NO\_STRETCH

***I2C own address2 masks***

I2C\_OA2\_NOMASK

I2C\_OA2\_MASK01

I2C\_OA2\_MASK02

I2C\_OA2\_MASK03

I2C\_OA2\_MASK04

I2C\_OA2\_MASK05

I2C\_OA2\_MASK06

I2C\_OA2\_MASK07

IS\_I2C\_OWN\_ADDRESS2\_MASK

***I2C Private Define***

TIMING\_CLEAR\_MASK

I2C\_TIMEOUT\_ADDR

I2C\_TIMEOUT\_BUSY

I2C\_TIMEOUT\_DIR

I2C\_TIMEOUT\_RXNE

I2C\_TIMEOUT\_STOPF

I2C\_TIMEOUT\_TC

I2C\_TIMEOUT\_TCR

I2C\_TIMEOUT\_TXIS

I2C\_TIMEOUT\_FLAG

***I2C ReloadEndMode definition***

I2C\_RELOAD\_MODE

I2C\_AUTOEND\_MODE

I2C\_SOFTEND\_MODE

IS\_TRANSFER\_MODE

***I2C StartStopMode definition***

I2C\_NO\_STARTSTOP

I2C\_GENERATE\_STOP

I2C\_GENERATE\_START\_READ

I2C\_GENERATE\_START\_WRITE

IS\_TRANSFER\_REQUEST

## 21 HAL I2C Extension Driver

### 21.1 I2CEEx Firmware driver API description

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

#### 21.1.1 I2C peripheral extension features

Comparing to other previous devices, the I2C interface for STM32F0XX devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop mode

#### 21.1.2 How to use this driver

This driver provides functions to configure Noise Filter

1. Configure I2C Analog noise filter using the function `HAL_I2CEEx_AnalogFilter_Config()`
2. Configure I2C Digital noise filter using the function `HAL_I2CEEx_DigitalFilter_Config()`
3. Configure the enabling or disabling of I2C Wake Up Mode using the functions :
  - `HAL_I2CEEx_EnableWakeUp()`
  - `HAL_I2CEEx_DisableWakeUp()`

#### 21.1.3 Extension features functions

This section provides functions allowing to:

- Configure Noise Filters
- `HAL_I2CEEx_AnalogFilter_Config()`
- `HAL_I2CEEx_DigitalFilter_Config()`
- `HAL_I2CEEx_EnableWakeUp()`
- `HAL_I2CEEx_DisableWakeUp()`

#### 21.1.4 `HAL_I2CEEx_AnalogFilter_Config`

Function Name	<code>HAL_StatusTypeDef HAL_I2CEEx_AnalogFilter_Config (</code> <code>I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)</code>
Function Description	Configures I2C Analog noise filter.
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 I2Cx peripheral.</li> <li>• <b>AnalogFilter</b> : new state of the Analog filter.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

## Notes

- None.

### 21.1.5 HAL\_I2CEx\_DigitalFilter\_Config

Function Name	<b>HAL_StatusTypeDef HAL_I2CEx_DigitalFilter_Config (</b> <b>I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)</b>
Function Description	Configures I2C Digital noise filter.
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 I2Cx peripheral.</li><li>• <b>DigitalFilter</b> : : Coefficient of digital noise filter between 0x00 and 0x0F.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 21.1.6 HAL\_I2CEx\_EnableWakeUp

Function Name	<b>HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (</b> <b>I2C_HandleTypeDef * hi2c)</b>
Function Description	Enables I2C wakeup from stop 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 I2Cx peripheral.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 21.1.7 HAL\_I2CEx\_DisableWakeUp

Function Name	<b>HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (</b>
---------------	----------------------------------------------------

***I2C\_HandleTypeDef \* hi2c)***

Function Description	Disables I2C wakeup from stop 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 I2Cx peripheral.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 21.2 I2CEEx Firmware driver defines

### 21.2.1 I2CEEx

I2CEEx

***I2CEEx Analog Filter***

I2C\_ANALOGFILTER\_ENABLED

I2C\_ANALOGFILTER\_DISABLED

IS\_I2C\_ANALOG\_FILTER

***I2CEEx Digital Filter***

IS\_I2C\_DIGITAL\_FILTER

## 22 HAL I2S Generic Driver

### 22.1 I2S Firmware driver registers structures

#### 22.1.1 I2S\_InitTypeDef

*I2S\_InitTypeDef* is defined in the `stm32f0xx_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 `stm32f0xx_hal_i2s.h`

##### Data Fields

- `SPI_TypeDef * Instance`
- `I2S_InitTypeDef Init`
- `uint16_t * pTxBuffPtr`
- `__IO uint16_t TxXferSize`
- `__IO uint16_t TxXferCount`
- `uint16_t * pRxBuffPtr`
- `__IO uint16_t RxXferSize`
- `__IO uint16_t RxXferCount`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `__IO HAL_LockTypeDef Lock`
- `__IO HAL_I2S_StateTypeDef State`
- `__IO HAL_I2S_ErrorTypeDef ErrorCode`

### Field Documentation

- *SPI\_TypeDef\* I2S\_HandleTypeDef::Instance*
- *I2S\_InitTypeDef I2S\_HandleTypeDef::Init*
- *uint16\_t\* I2S\_HandleTypeDef::pTxBuffPtr*
- *\_IO uint16\_t I2S\_HandleTypeDef::TxXferSize*
- *\_IO uint16\_t I2S\_HandleTypeDef::TxXferCount*
- *uint16\_t\* I2S\_HandleTypeDef::pRxBuffPtr*
- *\_IO uint16\_t I2S\_HandleTypeDef::RxXferSize*
- *\_IO uint16\_t I2S\_HandleTypeDef::RxXferCount*
- *DMA\_HandleTypeDef\* I2S\_HandleTypeDef::hdmatx*
- *DMA\_HandleTypeDef\* I2S\_HandleTypeDef::hdmarx*
- *\_IO HAL\_LockTypeDef I2S\_HandleTypeDef::Lock*
- *\_IO HAL\_I2S\_StateTypeDef I2S\_HandleTypeDef::State*
- *\_IO HAL\_I2S\_ErrorTypeDef I2S\_HandleTypeDef::ErrorCode*

## 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 pull-up.
  - c. NVIC configuration if you need to use interrupt process (HAL\_I2S\_Transmit\_IT() and HAL\_I2S\_Receive\_IT() APIs).
    - Configure the I2Sx interrupt priority.
    - Enable the NVIC I2S IRQ handle.
  - d. DMA Configuration if you need to use DMA process (HAL\_I2S\_Transmit\_DMA() and HAL\_I2S\_Receive\_DMA() APIs):
    - Declare a DMA handle structure for the Tx/Rx 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. Make sure that either: External clock source is configured after setting correctly the define constant EXTERNAL\_CLOCK\_VALUE in the stm32f0xx\_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

## 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_DelInit()` to restore the default configuration of the selected I2Sx peripheral.
- `HAL\_I2S\_Init\(\)`
- `HAL\_I2S\_DelInit\(\)`
- `HAL\_I2S\_MspInit\(\)`
- `HAL\_I2S\_MspDelInit\(\)`

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

#### 22.2.6 HAL\_I2S\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_I2S_DeInit ( <i>I2S_HandleTypeDef</i> *</code>
---------------	----------------------------------------------------------------------------

**hi2s)**

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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.7 HAL\_I2S\_MspInit

Function Name	<b>void HAL_I2S_MspInit ( <i>I2S_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.8 HAL\_I2S\_MspDeInit

Function Name	<b>void HAL_I2S_MspDeInit ( <i>I2S_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.9 HAL\_I2S\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit ( <i>I2S_HandleTypeDef</i> * <i>hi2s</i>, <i>uint16_t</i> * <i>pData</i>, <i>uint16_t</i> <i>Size</i>, <i>uint32_t</i> <i>Timeout</i>)</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> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></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 <b>Size</b> 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 <b>Size</b> 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>

## 22.2.10 HAL\_I2S\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive ( <i>I2S_HandleTypeDef</i> * <i>hi2s</i>, <i>uint16_t</i> * <i>pData</i>, <i>uint16_t</i> <i>Size</i>, <i>uint32_t</i> <i>Timeout</i>)</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> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></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 <b>Size</b> 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 <b>Size</b> 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 continuous 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 (</b> <b>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>• <b>HAL status</b></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> <b>I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>
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>• <b>HAL status</b></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).
- 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.

### 22.2.13 HAL\_I2S\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit_DMA (</b> <b>I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>
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>• <b>HAL status</b></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 <b>Size</b> 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 <b>Size</b> 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> <b>I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>
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><b>HAL status</b></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 (</b> <b>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><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 22.2.16 HAL\_I2S\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DMAResume (</b> <b>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><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 22.2.17 HAL\_I2S\_DMAMain

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DMAMain ( <i>I2S_HandleTypeDef</i> * <i>hi2s</i>)</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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 22.2.18 HAL\_I2S\_IRQHandler

Function Name	<b>void HAL_I2S_IRQHandler ( <i>I2S_HandleTypeDef</i> * <i>hi2s</i>)</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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 22.2.19 HAL\_I2S\_TxHalfCpltCallback

Function Name	<b>void HAL_I2S_TxHalfCpltCallback ( <i>I2S_HandleTypeDef</i> * <i>hi2s</i>)</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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.20 HAL\_I2S\_TxCpltCallback

Function Name	<b>void HAL_I2S_TxCpltCallback ( <i>I2S_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.21 HAL\_I2S\_RxHalfCpltCallback

Function Name	<b>void HAL_I2S_RxHalfCpltCallback ( <i>I2S_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.22 HAL\_I2S\_RxCpltCallback

Function Name	<b>void HAL_I2S_RxCpltCallback ( <i>I2S_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.23 HAL\_I2S\_ErrorCallback

Function Name	<b>void HAL_I2S_ErrorCallback ( <i>I2S_HandleTypeDef</i> * 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>
Notes	<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 ( <i>I2S_HandleTypeDef</i> * 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>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.25 HAL\_I2S\_GetError

Function Name	<b>HAL_I2S_ErrorTypeDef HAL_I2S_GetError ( <i>I2S_HandleTypeDef</i> * 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>• <b>I2S Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 22.3 I2S Firmware driver defines

### 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  
IS\_I2S\_AUDIO\_FREQ

#### *I2S Clock Polarity*

I2S\_CPOL\_LOW  
I2S\_CPOL\_HIGH  
IS\_I2S\_CPOL

#### *I2S Data Format*

I2S\_DATAFORMAT\_16B  
I2S\_DATAFORMAT\_16B\_EXTENDED  
I2S\_DATAFORMAT\_24B  
I2S\_DATAFORMAT\_32B  
IS\_I2S\_DATA\_FORMAT

#### *I2S Exported Macros*

<code>_HAL_I2S_RESET_HANDLE_STATE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Reset I2S handle state.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>_HANDLE_</code>: I2S handle.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None:</li></ul>
<code>_HAL_I2S_ENABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Enable or disable the specified SPI</li></ul>

peripheral (in I2S mode).

**Parameters:**

- `__HANDLE__`: specifies the I2S Handle.

**Return value:**

- None:

`__HAL_I2S_DISABLE`

`__HAL_I2S_ENABLE_IT`

**Description:**

- Enable or 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_DISABLE_IT`

`__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 `I2Sx` 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).

**Description:**

- Checks whether the specified I2S flag is set or not.

**Parameters:**

- `__HANDLE__`: specifies the I2S Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `I2S_FLAG_RXNE`: Receive buffer not empty flag
  - `I2S_FLAG_TXE`: Transmit buffer empty flag
  - `I2S_FLAG_UDR`: Underrun flag
  - `I2S_FLAG_OVR`: Overrun flag
  - `I2S_FLAG_FRE`: Frame error flag
  - `I2S_FLAG_CHSIDE`: Channel Side flag
  - `I2S_FLAG_BSY`: Busy flag

**Return value:**

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

`__HAL_I2S_CLEAR_OVRFAG`**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  
IS\_I2S\_MCLK\_OUTPUT

***I2S Mode***

I2S\_MODE\_SLAVE\_TX  
I2S\_MODE\_SLAVE\_RX  
I2S\_MODE\_MASTER\_TX  
I2S\_MODE\_MASTER\_RX  
IS\_I2S\_MODE

***I2S Standard***

I2S\_STANDARD\_PHILIPS  
I2S\_STANDARD\_MSB  
I2S\_STANDARD\_LSB  
I2S\_STANDARD\_PCM\_SHORT  
I2S\_STANDARD\_PCM\_LONG  
IS\_I2S\_STANDARD

## 23 HAL IRDA Generic Driver

### 23.1 IRDA Firmware driver registers structures

#### 23.1.1 IRDA\_InitTypeDef

*IRDA\_InitTypeDef* is defined in the stm32f0xx\_hal\_irda.h

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint16\_t Parity*
- *uint16\_t Mode*
- *uint8\_t Prescaler*
- *uint16\_t PowerMode*

##### Field Documentation

- ***uint32\_t IRDA\_InitTypeDef::BaudRate*** This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hirda->Init.BaudRate)))
- ***uint32\_t IRDA\_InitTypeDef::WordLength*** Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [\*IRDAEx\\_Word\\_Length\*](#)
- ***uint16\_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).
- ***uint16\_t IRDA\_InitTypeDef::Mode*** Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [\*IRDA\\_Mode\*](#)
- ***uint8\_t IRDA\_InitTypeDef::Prescaler*** Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.  
**Note:**Prescaler value 0 is forbidden
- ***uint16\_t IRDA\_InitTypeDef::PowerMode*** Specifies the IRDA power mode. This parameter can be a value of [\*IRDA\\_Low\\_Power\*](#)

#### 23.1.2 IRDA\_HandleTypeDef

*IRDA\_HandleTypeDef* is defined in the stm32f0xx\_hal\_irda.h

##### Data Fields

- *USART\_TypeDef \* Instance*
- *IRDA\_InitTypeDef Init*
- *uint8\_t \* pTxBuffPtr*
- *uint16\_t TxXferSize*
- *uint16\_t TxXferCount*
- *uint8\_t \* pRxBuffPtr*
- *uint16\_t RxXferSize*
- *uint16\_t RxXferCount*

- *uint16\_t Mask*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*
- *HAL\_LockTypeDef Lock*
- *HAL\_IRDA\_StateTypeDef State*
- *HAL\_IRDA\_ErrorTypeDef ErrorCode*

#### Field Documentation

- *USART\_TypeDef\* IRDA\_HandleTypeDef::Instance*
- *IRDA\_InitTypeDef IRDA\_HandleTypeDef::Init*
- *uint8\_t\* IRDA\_HandleTypeDef::pTxBuffPtr*
- *uint16\_t IRDA\_HandleTypeDef::TxXferSize*
- *uint16\_t IRDA\_HandleTypeDef::TxXferCount*
- *uint8\_t\* IRDA\_HandleTypeDef::pRxBuffPtr*
- *uint16\_t IRDA\_HandleTypeDef::RxXferSize*
- *uint16\_t IRDA\_HandleTypeDef::RxXferCount*
- *uint16\_t IRDA\_HandleTypeDef::Mask*
- *DMA\_HandleTypeDef\* IRDA\_HandleTypeDef::hdmatx*
- *DMA\_HandleTypeDef\* IRDA\_HandleTypeDef::hdmarx*
- *HAL\_LockTypeDef IRDA\_HandleTypeDef::Lock*
- *HAL\_IRDA\_StateTypeDef IRDA\_HandleTypeDef::State*
- *HAL\_IRDA\_ErrorTypeDef IRDA\_HandleTypeDef::ErrorCode*

## 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 these IRDA pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (HAL\_IRDA\_Transmit\_IT() and HAL\_IRDA\_Receive\_IT() APIs):
    - Configure the USARTx interrupt priority.
    - Enable the NVIC USART IRQ handle.
  - d. DMA Configuration if you need to use DMA process (HAL\_IRDA\_Transmit\_DMA() and HAL\_IRDA\_Receive\_DMA() APIs):
    - Declare a DMA handle structure for the Tx/Rx 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.
3. Program the Baud Rate, Word Length and Parity and Mode(Receiver/Transmitter), the normal or low power mode and the clock prescaler 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



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 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) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit), the possible IRDA frame formats are as listed in the following table
  - Power mode
  - Prescaler setting
  - Receiver/transmitter modes

**Table 19: 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
M1, M0 bits	PCE bit	IRDA frame
10	0	SB   7-bit data   STB
10	1	SB   6-bit data   PB   STB

The HAL\_IRDA\_Init() function follows IRDA configuration procedures (details for the procedures are available in reference manual).

- [\*\*HAL\\_IRDA\\_Init\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_MspDelInit\(\)\*\*](#)

### 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.
  - Non Blocking mode: The communication is performed using Interrupts or DMA, these API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_IRDA\_TxCpltCallback(), HAL\_IRDA\_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The HAL\_IRDA\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode API's are :
    - HAL\_IRDA\_Transmit()
    - HAL\_IRDA\_Receive()
  3. Non Blocking mode API's with Interrupt are :
    - HAL\_IRDA\_Transmit\_IT()
    - HAL\_IRDA\_Receive\_IT()
    - HAL\_IRDA\_IRQHandler()
    - IRDA\_Transmit\_IT()
    - IRDA\_Receive\_IT()
  4. Non Blocking mode functions with DMA are :
    - HAL\_IRDA\_Transmit\_DMA()
    - HAL\_IRDA\_Receive\_DMA()
  5. A set of Transfer Complete Callbacks are provided in Non Blocking mode:
    - HAL\_IRDA\_TxCpltCallback()
    - HAL\_IRDA\_RxCpltCallback()
    - HAL\_IRDA\_ErrorCallback()
    - [\*\*HAL\\_IRDA\\_Transmit\(\)\*\*](#)
    - [\*\*HAL\\_IRDA\\_Receive\(\)\*\*](#)
    - [\*\*HAL\\_IRDA\\_Transmit\\_IT\(\)\*\*](#)
    - [\*\*HAL\\_IRDA\\_Receive\\_IT\(\)\*\*](#)
    - [\*\*HAL\\_IRDA\\_Transmit\\_DMA\(\)\*\*](#)
    - [\*\*HAL\\_IRDA\\_Receive\\_DMA\(\)\*\*](#)
    - [\*\*HAL\\_IRDA\\_IRQHandler\(\)\*\*](#)
    - [\*\*HAL\\_IRDA\\_TxCpltCallback\(\)\*\*](#)
    - [\*\*HAL\\_IRDA\\_RxCpltCallback\(\)\*\*](#)
    - [\*\*HAL\\_IRDA\\_ErrorCallback\(\)\*\*](#)

### 23.2.4 Peripheral Control functions

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

- HAL\_IRDA\_GetState() API can be helpful to check in run-time the state of the IRDA peripheral.
- IRDA\_SetConfig() API is used to configure the IRDA communications parameters.
- [\*\*HAL\\_IRDA\\_GetState\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_GetError\(\)\*\*](#)

### 23.2.5 HAL\_IRDA\_Init

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Init ( <a href="#"><b>IRDA_HandleTypeDef</b></a> * *</b>
---------------	----------------------------------------------------------------------------------------

**hirda)**

Function Description	Initializes the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and creates the associated handle .
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b> : IRDA handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 23.2.6 HAL\_IRDA\_DelInit

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

### 23.2.7 HAL\_IRDA\_MspInit

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

### 23.2.8 HAL\_IRDA\_MspDelInit

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

---

Function Description	IRDA MSP Delinit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : IRDA handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.9 HAL\_IRDA\_Transmit

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

### 23.2.10 HAL\_IRDA\_Receive

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

### 23.2.11 HAL\_IRDA\_Transmit\_IT

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

### 23.2.12 HAL\_IRDA\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive_IT (</b> <b><i>IRDA_HandleTypeDef</i> * hirda, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : IRDA handle</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.13 HAL\_IRDA\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (</b> <b><i>IRDA_HandleTypeDef</i> * hirda, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : IRDA handle</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>• <b>HAL status</b></li> </ul>

## Notes

- None.

### 23.2.14 HAL\_IRDA\_Receive\_DMA

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

### 23.2.15 HAL\_IRDA\_IRQHandler

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

### 23.2.16 HAL\_IRDA\_TxCpltCallback

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

---

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

### 23.2.17 HAL\_IRDA\_RxCpltCallback

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

### 23.2.18 HAL\_IRDA\_ErrorCallback

Function Name	<b>void HAL_IRDA_ErrorCallback ( <i>IRDA_HandleTypeDef</i> * hirda)</b>
Function Description	IRDA error callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b> : IRDA handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 23.2.19 HAL\_IRDA\_GetState

Function Name	<b>HAL_IRDA_StateTypeDef HAL_IRDA_GetState ( <i>IRDA_HandleTypeDef</i> * hirda)</b>
Function Description	return the IRDA state
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b> : irda handle</li></ul>

---

Return values	<ul style="list-style-type: none"> <li><b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 23.2.20 HAL\_IRDA\_GetError

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

## 23.3 IRDA Firmware driver defines

### 23.3.1 IRDA

IRDA

***IRDA DMA Rx***

IRDA\_DMA\_RX\_DISABLE

IRDA\_DMA\_RX\_ENABLE

IS\_IRDA\_DMA\_RX

***IRDA DMA Tx***

IRDA\_DMA\_TX\_DISABLE

IRDA\_DMA\_TX\_ENABLE

IS\_IRDA\_DMA\_TX

***IRDA Exported Macros***

***\_HAL\_IRDA\_RESET\_HANDLE\_STA*** **Description:**

- Reset IRDA handle state.

**Parameters:**

- \_HANDLE***: IRDA handle.

**Return value:**

- None:

[\\_\\_HAL\\_IRDA\\_GET\\_FLAG](#)**Description:**

- Checks whether the specified IRDA flag is set or not.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- [\\_\\_FLAG\\_\\_](#): specifies the flag to check. This parameter can be one of the following values:
  - IRDA\_FLAG\_RXACK: Receive enable acknowledge flag
  - IRDA\_FLAG\_TEACK: Transmit enable acknowledge flag
  - IRDA\_FLAG\_BUSY: Busy flag
  - IRDA\_FLAG\_ABRF: Auto Baud rate detection flag
  - IRDA\_FLAG\_ABRE: Auto Baud rate detection error flag
  - IRDA\_FLAG\_TXE: Transmit data register empty flag
  - IRDA\_FLAG\_TC: Transmission Complete flag
  - IRDA\_FLAG\_RXNE: Receive data register not empty flag
  - IRDA\_FLAG\_IDLE: Idle Line detection flag
  - IRDA\_FLAG\_ORE: OverRun Error flag
  - IRDA\_FLAG\_NE: Noise Error flag
  - IRDA\_FLAG\_FE: Framing Error flag
  - IRDA\_FLAG\_PE: Parity Error flag

**Return value:**

- The new state of [\\_\\_FLAG\\_\\_](#) (TRUE or FALSE).

[\\_\\_HAL\\_IRDA\\_ENABLE\\_IT](#)**Description:**

- Enables the specified IRDA interrupt.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- [\\_\\_INTERRUPT\\_\\_](#): specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data register

- not empty interrupt
- IRDA\_IT\_IDLE: Idle line detection interrupt
- IRDA\_IT\_PE: Parity Error interrupt
- IRDA\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

**\_HAL\_IRDA\_DISABLE\_IT**

- Disables the specified IRDA interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance can be USARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- \_\_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**

- Checks whether the specified IRDA interrupt has occurred or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance can be USARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- \_\_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\_ORE: OverRun Error interrupt
- IRDA\_IT\_NE: Noise Error interrupt
- IRDA\_IT\_FE: Framing Error interrupt
- IRDA\_IT\_PE: Parity Error interrupt

**Return value:**

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

\_HAL\_IRDA\_GET\_IT\_SOURCE**Description:**

- Checks whether the specified IRDA interrupt source is enabled.

**Parameters:**

- HANDLE: specifies the IRDA Handle. The Handle Instance can be USARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- 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\_ORE: OverRun Error interrupt
  - IRDA\_IT\_NE: Noise Error interrupt
  - IRDA\_IT\_FE: Framing Error interrupt
  - IRDA\_IT\_PE: Parity Error interrupt

**Return value:**

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

\_HAL\_IRDA\_CLEAR\_IT**Description:**

- Clears the specified IRDA ISR flag, in setting the proper ICR register flag.

**Parameters:**

- HANDLE: specifies the IRDA Handle. The Handle Instance can be USARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- IT\_CLEAR: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - IRDA\_CLEAR\_PEF: Parity Error Clear Flag
  - IRDA\_CLEAR\_FEF: Framing Error

- Clear Flag
- IRDA\_CLEAR\_NEF: Noise detected
- Clear Flag
- IRDA\_CLEAR\_OREF: OverRun Error
- Clear Flag
- IRDA\_CLEAR\_TCF: Transmission Complete Clear Flag

**Return value:**

- None:

**\_HAL\_IRDA\_SEND\_REQ**

- Set a specific IRDA request flag.

**Parameters:**

- \_HANDLE\_: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- \_REQ\_: specifies the request flag to set. This parameter can be one of the following values:
  - IRDA\_AUTOBAUD\_REQUEST: Auto-Baud Rate Request
  - IRDA\_RXDATA\_FLUSH\_REQUEST: Receive Data flush Request
  - IRDA\_TXDATA\_FLUSH\_REQUEST: Transmit data flush Request

**Return value:**

- None:

**\_HAL\_IRDA\_ENABLE**

- Enable UART/USART associated to IRDA Handle.

**Parameters:**

- \_HANDLE\_: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral

**Return value:**

- None:

**\_HAL\_IRDA\_DISABLE**

- Disable UART/USART associated to IRDA Handle.

**Parameters:**

- \_HANDLE\_: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral

**Return value:**

- None:

**IS\_IRDA\_BAUDRATE****Description:**

- Ensure that IRDA Baud rate is less or equal to maximum value.

**Parameters:**

- \_\_BAUDRATE\_\_: specifies the IRDA Baudrate set by the user.

**Return value:**

- True: or False

**IS\_IRDA\_PRESCALER****Description:**

- Ensure that IRDA prescaler value is strictly larger than 0.

**Parameters:**

- \_\_PRESCALER\_\_: specifies the IRDA prescaler value set by the user.

**Return value:**

- True: or False

***IRDA Flags*****IRDA\_FLAG\_RXACK****IRDA\_FLAG\_TEACK****IRDA\_FLAG\_BUSY****IRDA\_FLAG\_ABRF****IRDA\_FLAG\_ABRE****IRDA\_FLAG\_TXE****IRDA\_FLAG\_TC****IRDA\_FLAG\_RXNE****IRDA\_FLAG\_ORE****IRDA\_FLAG\_NE****IRDA\_FLAG\_FE****IRDA\_FLAG\_PE*****IRDA interruptions flag mask*****IRDA\_IT\_MASK*****IRDA Interrupt Definition*****IRDA\_IT\_PE****IRDA\_IT\_TXE****IRDA\_IT\_TC**

IRDA\_IT\_RXNE

IRDA\_IT\_IDLE

IRDA\_IT\_ERR

IRDA\_IT\_ORE

IRDA\_IT\_NE

IRDA\_IT\_FE

***IRDA Interruption Clear Flags***

IRDA\_CLEAR\_PEF      Parity Error Clear Flag

IRDA\_CLEAR\_FEF      Framing Error Clear Flag

IRDA\_CLEAR\_NEF      Noise detected Clear Flag

IRDA\_CLEAR\_OREF      OverRun Error Clear Flag

IRDA\_CLEAR\_TCF      Transmission Complete Clear Flag

***IRDA Low Power***

IRDA\_POWERMODE\_NORMAL

IRDA\_POWERMODE\_LOWPOWER

IS\_IRDA\_POWERMODE

***IRDA Mode***

IRDA\_MODE\_DISABLE

IRDA\_MODE\_ENABLE

IS\_IRDA\_MODE

***IRDA One Bit Sampling***

IRDA\_ONE\_BIT\_SAMPLE\_DISABLED

IRDA\_ONE\_BIT\_SAMPLE\_ENABLED

IS\_IRDA\_ONEBIT\_SAMPLE

***IRDA Parity***

IRDA\_PARITY\_NONE

IRDA\_PARITY\_EVEN

IRDA\_PARITY\_ODD

IS\_IRDA\_PARITY

***IRDA Private Constants***

TEACK\_REACK\_TIMEOUT

IRDA\_TXDMA\_TIMEOUTVALUE

IRDA\_TIMEOUT\_VALUE

IRDA\_CR1\_FIELDS

***IRDA Request Parameters***

IRDA\_AUTOBAUD\_REQUEST      Auto-Baud Rate Request

IRDA\_RXDATA\_FLUSH\_REQUEST Receive Data flush Request

IRDA\_TXDATA\_FLUSH\_REQUEST Transmit data flush Request

IS\_IRDA\_REQUEST\_PARAMETER

***IRDA State***

IRDA\_STATE\_DISABLE

IRDA\_STATE\_ENABLE

IS\_IRDA\_STATE

***IRDA Transfer Mode***

IRDA\_MODE\_RX

IRDA\_MODE\_TX

IRDA\_MODE\_TX\_RX

IS\_IRDA\_TX\_RX\_MODE

## 24 HAL IRDA Extension Driver

### 24.1 IRDAEx Firmware driver defines

#### 24.1.1 IRDAEx

IRDAEx

*IRDAEx Exported Macros*

`_HAL_IRDA_GETCLOCKSOURCE`

**Description:**

- Reports the IRDA clock source.

**Parameters:**

- `_HANDLE_`: specifies the IRDA Handle
- `_CLOCKSOURCE_`: output variable

**Return value:**

- IRDA: clocking source, written in `_CLOCKSOURCE_`.

`_HAL_IRDA_MASK_COMPUTATION`

**Description:**

- Computes the mask to apply to retrieve the received data according to the word length and to the parity bits activation.

**Parameters:**

- `_HANDLE_`: specifies the IRDA Handle

**Return value:**

- none:

*IRDA Word Length*

`IRDA_WORDLENGTH_7B`

`IRDA_WORDLENGTH_8B`

`IRDA_WORDLENGTH_9B`

`IS_IRDA_WORD_LENGTH`

## 25 HAL IWDG Generic Driver

### 25.1 IWDG Firmware driver registers structures

#### 25.1.1 IWDG\_InitTypeDef

*IWDG\_InitTypeDef* is defined in the `stm32f0xx_hal_iwdg.h`

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Reload*
- *uint32\_t Window*

##### Field Documentation

- *uint32\_t IWDG\_InitTypeDef::Prescaler* Select the prescaler of the IWDG. This parameter can be a value of *IWDG\_Prescaler*
- *uint32\_t IWDG\_InitTypeDef::Reload* Specifies the IWDG down-counter reload value. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x0FFF
- *uint32\_t IWDG\_InitTypeDef::Window* Specifies the window value to be compared to the down-counter. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x0FFF

#### 25.1.2 IWDG\_HandleTypeDefDef

*IWDG\_HandleTypeDefDef* is defined in the `stm32f0xx_hal_iwdg.h`

##### Data Fields

- *IWDG\_TypeDef \* Instance*
- *IWDG\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_IWDG\_StateTypeDef State*

##### Field Documentation

- *IWDG\_TypeDef\* IWDG\_HandleTypeDefDef::Instance* Register base address
- *IWDG\_InitTypeDef IWDG\_HandleTypeDefDef::Init* IWDG required parameters
- *HAL\_LockTypeDef IWDG\_HandleTypeDefDef::Lock* IWDG Locking object
- *\_\_IO HAL\_IWDG\_StateTypeDef IWDG\_HandleTypeDefDef::State* IWDG communication state

### 25.2 IWDG Firmware driver API description

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

#### 25.2.1 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 @41KHz (LSI): ~0.1ms / ~25.5s The IWDG timeout may vary due to LSI frequency dispersion. STM32F0x devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy. For more information, please refer to the STM32F0x Reference manual.

## 25.2.2 How to use this driver

1. if Window option is disabled
  - Use IWDG using HAL\_IWDG\_Init() function to :
    - Enable write access to IWDG\_PR, IWDG\_RLR.
    - Configure the IWDG prescaler, counter reload value. This reload value will be loaded in the IWDG counter each time the counter is reloaded, then the IWDG will start counting down from this value.
  - Use IWDG using HAL\_IWDG\_Start() function to :
    - Reload IWDG counter with value defined in the IWDG\_RLR register.
    - Start the IWDG, when the IWDG is used in software mode (no need to enable the LSI, it will be enabled by hardware).
  - Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL\_IWDG\_Refresh() function.
2. if Window option is enabled:
  - Use IWDG using HAL\_IWDG\_Start() function to enable IWDG downcounter
  - Use IWDG using HAL\_IWDG\_Init() function to :
    - Enable write access to IWDG\_PR, IWDG\_RLR and IWDG\_WINR registers.
    - Configure the IWDG prescaler, reload value and window value.
  - Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL\_IWDG\_Refresh() function.

### IWDG HAL driver macros list

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

- \_\_HAL\_IWDG\_START: Enable the IWDG peripheral
- \_\_HAL\_IWDG\_RELOAD\_COUNTER: Reloads IWDG counter with value defined in the reload register

- `_HAL_IWDG_ENABLE_WRITE_ACCESS` : Enable write access to IWDG\_PR and IWDG\_RLR registers
- `_HAL_IWDG_DISABLE_WRITE_ACCESS` : Disable write access to IWDG\_PR and IWDG\_RLR registers
- `_HAL_IWDG_GET_FLAG`: Get the selected IWDG's flag status

### 25.2.3 Initialization functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG\_InitTypeDef and create the associated handle
- Manage Window option
- Initialize the IWDG MSP
- Deinitialize IWDG MSP
- `HAL_IWDG_Init()`
- `HAL_IWDG_MspInit()`

### 25.2.4 IO operation functions

This section provides functions allowing to:

- Start the IWDG.
- Refresh the IWDG.
- `HAL_IWDG_Start()`
- `HAL_IWDG_Refresh()`

### 25.2.5 Peripheral State functions

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

- `HAL_IWDG_GetState()`

### 25.2.6 HAL\_IWDG\_Init

Function Name	<code>HAL_StatusTypeDef HAL_IWDG_Init ( IWDG_HandleTypeDef * hiwdg)</code>
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>• <code>hiwdg</code> : 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 25.2.7 HAL\_IWDG\_MspInit

Function Name	<b>void HAL_IWDG_MspInit ( <i>IWDG_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 25.2.8 HAL\_IWDG\_Start

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Start ( <i>IWDG_HandleTypeDef</i> * 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	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 25.2.9 HAL\_IWDG\_Refresh

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Refresh ( <i>IWDG_HandleTypeDef</i> * hiwdg)</b>
Function Description	Refreshes 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	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>

## Notes

- None.

### 25.2.10 HAL\_IWDG\_GetState

Function Name	<b>HAL_IWDG_StateTypeDef HAL_IWDG_GetState (</b> <b>IWDG_HandleTypeDef * hiwdg)</b>
Function Description	Returns the IWDG state.
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>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 25.3 IWDG Firmware driver defines

### 25.3.1 IWDG

IWDG

*IWDG CounterWindow Value*

IS\_IWDG\_WINDOW

*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:**

- None:

[\\_\\_HAL\\_IWDG\\_RELOAD\\_COUNTER](#)**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\\_ENABLE\\_WRITE\\_ACCESS](#)**Description:**

- Enable write access to IWDG\_PR, IWDG\_RLR and IWDG\_WINR registers.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): IWDG handle

**Return value:**

- None:

[\\_\\_HAL\\_IWDG\\_DISABLE\\_WRITE\\_ACCESS](#)**Description:**

- Disable write access to IWDG\_PR, IWDG\_RLR and IWDG\_WINR registers.

**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
  - IWDG\_FLAG\_WVU: Watchdog counter window 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_FLAG_WVU	Watchdog counter window 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
IS_IWDG_PRESCALER	

***IWDG Private Defines***

HAL\_IWDG\_DEFAULT\_TIMEOUT

***IWDG Registers BitMask***

KR_KEY_RELOAD	IWDG Reload Counter Enable
KR_KEY_ENABLE	IWDG Peripheral Enable
KR_KEY_EWA	IWDG KR Write Access Enable
KR_KEY_DWA	IWDG KR Write Access Disable
IS_IWDG_KR	

***IWDG Reload Value***

IS\_IWDG\_RELOAD

***IWDG Window option***

IWDG\_WINDOW\_DISABLE

## 26 HAL PCD Generic Driver

### 26.1 PCD Firmware driver registers structures

#### 26.1.1 PCD\_InitTypeDef

*PCD\_InitTypeDef* is defined in the stm32f0xx\_hal\_pcd.h

##### Data Fields

- *uint32\_t dev\_endpoints*
- *uint32\_t speed*
- *uint32\_t ep0\_mps*
- *uint32\_t phy\_iface*
- *uint32\_t Sof\_enable*
- *uint32\_t low\_power\_enable*
- *uint32\_t lpm\_enable*
- *uint32\_t battery\_charging\_enable*

##### Field Documentation

- ***uint32\_t PCD\_InitTypeDef::dev\_endpoints*** Device Endpoints number. This parameter depends on the used USB core. This parameter must be a number between Min\_Data = 1 and Max\_Data = 15
- ***uint32\_t PCD\_InitTypeDef::speed*** USB Core speed. This parameter can be any value of [PCD\\_Core\\_Speed](#)
- ***uint32\_t PCD\_InitTypeDef::ep0\_mps*** Set the Endpoint 0 Max Packet size. This parameter can be any value of [PCD\\_EP0\\_MPS](#)
- ***uint32\_t PCD\_InitTypeDef::phy\_iface*** Select the used PHY interface. This parameter can be any value of [PCD\\_Core\\_PHY](#)
- ***uint32\_t PCD\_InitTypeDef::Sof\_enable*** Enable or disable the output of the SOF signal. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t PCD\_InitTypeDef::low\_power\_enable*** Enable or disable Low Power mode. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t PCD\_InitTypeDef::lpm\_enable*** Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE
- ***uint32\_t PCD\_InitTypeDef::battery\_charging\_enable*** Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE

#### 26.1.2 PCD\_EPTypeDef

*PCD\_EPTypeDef* is defined in the stm32f0xx\_hal\_pcd.h

##### Data Fields

- *uint8\_t num*
- *uint8\_t is\_in*
- *uint8\_t is\_stall*
- *uint8\_t type*
- *uint16\_t pmaaddress*
- *uint16\_t pmaaddr0*
- *uint16\_t pmaaddr1*

- *uint8\_t doublebuffer*
- *uint32\_t maxpacket*
- *uint8\_t \* xfer\_buff*
- *uint32\_t xfer\_len*
- *uint32\_t xfer\_count*

#### Field Documentation

- *uint8\_t PCD\_EPTypedef::num* Endpoint number This parameter must be a number between Min\_Data = 1 and Max\_Data = 15
- *uint8\_t PCD\_EPTypedef::is\_in* Endpoint direction This parameter must be a number between Min\_Data = 0 and Max\_Data = 1
- *uint8\_t PCD\_EPTypedef::is\_stall* Endpoint stall condition This parameter must be a number between Min\_Data = 0 and Max\_Data = 1
- *uint8\_t PCD\_EPTypedef::type* Endpoint type This parameter can be any value of **PCD\_EP\_Type**
- *uint16\_t PCD\_EPTypedef::pmaaddress* PMA Address This parameter can be any value between Min\_addr = 0 and Max\_addr = 1K
- *uint16\_t PCD\_EPTypedef::pmaaddr0* PMA Address0 This parameter can be any value between Min\_addr = 0 and Max\_addr = 1K
- *uint16\_t PCD\_EPTypedef::pmaaddr1* PMA Address1 This parameter can be any value between Min\_addr = 0 and Max\_addr = 1K
- *uint8\_t PCD\_EPTypedef::doublebuffer* Double buffer enable This parameter can be 0 or 1
- *uint32\_t PCD\_EPTypedef::maxpacket* Endpoint Max packet size This parameter must be a number between Min\_Data = 0 and Max\_Data = 64KB
- *uint8\_t\* PCD\_EPTypedef::xfer\_buff* Pointer to transfer buffer
- *uint32\_t PCD\_EPTypedef::xfer\_len* Current transfer length
- *uint32\_t PCD\_EPTypedef::xfer\_count* Partial transfer length in case of multi packet transfer

### 26.1.3 PCD\_HandleTypeDefDef

**PCD\_HandleTypeDefDef** is defined in the stm32f0xx\_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\_HandleTypeDefDef::Instance* Register base address
- *PCD\_InitTypeDef PCD\_HandleTypeDefDef::Init* PCD required parameters
- *\_IO uint8\_t PCD\_HandleTypeDefDef::USB\_Address* USB Address
- *PCD\_EPTypedef PCD\_HandleTypeDefDef::IN\_ep[8]* IN endpoint parameters

- ***PCD\_EPTTypeDef PCD\_HandleTypeDef::OUT\_ep[8]*** OUT endpoint parameters
- ***HAL\_LockTypeDef PCD\_HandleTypeDef::Lock*** PCD peripheral status
- ***\_IO PCD\_StatusTypeDef PCD\_HandleTypeDef::State*** PCD communication state
- ***uint32\_t PCD\_HandleTypeDef::Setup[12]*** Setup packet buffer
- ***void\* PCD\_HandleTypeDef::pData*** Pointer to upper stack Handler

## 26.2 PCD Firmware driver API description

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

### 26.2.1 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  
– \_\_USB\_CLK\_ENABLE);
  - b. Initialize the related GPIO clocks
  - c. Configure PCD pin-out
  - d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
  - a. hpcd.pData = pdev;
6. Enable HCD transmission and reception:
  - a. HAL\_PCD\_Start();

### 26.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- ***HAL\_PCD\_Init()***
- ***HAL\_PCD\_Delinit()***
- ***HAL\_PCD\_MspInit()***
- ***HAL\_PCD\_MspDelinit()***

#### 26.2.2.1 HAL\_PCD\_Init

Function Name	<b><i>HAL_StatusTypeDef HAL_PCD_Init ( PCD_HandleTypeDef * hpcd)</i></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><i>hpcd</i></b> : PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b><i>HAL status</i></b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 26.2.3 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()*

### 26.2.4 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()*

### 26.2.5 Peripheral State functions

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

- *HAL\_PCD\_GetState()*

### 26.2.6 HAL\_PCD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Init ( <i>PCD_HandleTypeDef</i> * <i>hpcd</i>)</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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.7 HAL\_PCD\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DelInit ( <i>PCD_HandleTypeDef</i> * <i>hpcd</i>)</b>
Function Description	Delinitializes 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.8 HAL\_PCD\_MspInit

Function Name	<b>void HAL_PCD_MspInit ( <i>PCD_HandleTypeDef</i> * <i>hpcd</i>)</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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.9 HAL\_PCD\_MspDelInit

---

Function Name	<b>void HAL_PCD_MspDelInit ( <i>PCD_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.10 HAL\_PCD\_Start

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Start ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Start The USB OTG 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.11 HAL\_PCD\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Stop ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Stop The USB OTG 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.12 HAL\_PCD\_IRQHandler

Function Name	<b>void HAL_PCD_IRQHandler ( <i>PCD_HandleTypeDef</i> * 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.13 HAL\_PCD\_DataOutStageCallback

Function Name	<b>void HAL_PCD_DataOutStageCallback ( <i>PCD_HandleTypeDef</i> * <b>hpcd</b>, uint8_t <b>epnum</b>)</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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.14 HAL\_PCD\_DataInStageCallback

Function Name	<b>void HAL_PCD_DataInStageCallback ( <i>PCD_HandleTypeDef</i> * <b>hpcd</b>, uint8_t <b>epnum</b>)</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	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.15 HAL\_PCD\_SetupStageCallback

Function Name	<b>void HAL_PCD_SetupStageCallback ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Setup stage callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : ppp handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.16 HAL\_PCD\_SOFCallback

Function Name	<b>void HAL_PCD_SOFCallback ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	USB Start Of Frame callbacks.
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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.17 HAL\_PCD\_ResetCallback

Function Name	<b>void HAL_PCD_ResetCallback ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	USB Reset callbacks.
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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.18 HAL\_PCD\_SuspendCallback

Function Name	<b>void HAL_PCD_SuspendCallback ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
---------------	------------------------------------------------------------------------

Function Description	Suspend event callbacks.
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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 26.2.19 HAL\_PCD\_ResumeCallback

Function Name	<b>void HAL_PCD_ResumeCallback ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Resume event callbacks.
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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 26.2.20 HAL\_PCD\_ISOOUTIncompleteCallback

Function Name	<b>void HAL_PCD_ISOOUTIncompleteCallback ( <i>PCD_HandleTypeDef</i> * hpcd, uint8_t epi)</b>
Function Description	Incomplete ISO OUT callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li><li>• <b>epi</b> : endpoint number</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 26.2.21 HAL\_PCD\_ISOINIncompleteCallback

Function Name	<b>void HAL_PCD_ISOINIncompleteCallback (</b>
---------------	-----------------------------------------------

***PCD\_HandleTypeDefDef \* hpcd, uint8\_t epcnum)***

Function Description	Incomplete ISO IN callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li><li>• <b>epcnum</b> : endpoint number</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.22 HAL\_PCD\_ConnectCallback

Function Name	<b>void HAL_PCD_ConnectCallback ( <i>PCD_HandleTypeDefDef *</i> hpcd)</b>
Function Description	Connection event callbacks.
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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.23 HAL\_PCD\_DisconnectCallback

Function Name	<b>void HAL_PCD_DisconnectCallback ( <i>PCD_HandleTypeDefDef *</i> hpcd)</b>
Function Description	Disconnection event callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : ppp handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.24 HAL\_PCD\_DevConnect

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DevConnect (</b> <b><i>PCD_HandleTypeDef * hpcd</i></b> )
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.25 HAL\_PCD\_DevDisconnect

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DevDisconnect (</b> <b><i>PCD_HandleTypeDef * hpcd</i></b> )
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.26 HAL\_PCD\_SetAddress

Function Name	<b>HAL_StatusTypeDef HAL_PCD_SetAddress (</b> <b><i>PCD_HandleTypeDef * hpcd, uint8_t address</i></b> )
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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.27 HAL\_PCD\_EP\_Open

---

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Open (</b> <b>PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t</b> <b>ep_mps, uint8_t ep_type)</b>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 26.2.28 HAL\_PCD\_EP\_Close

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Close (</b> <b>PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
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	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 26.2.29 HAL\_PCD\_EP\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Receive (</b> <b>PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf,</b> <b>uint32_t len)</b>
Function Description	Receive an amount of data.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> <li>• <b>ep_addr</b> : endpoint address</li> <li>• <b>pBuf</b> : pointer to the reception buffer</li> <li>• <b>len</b> : amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 26.2.30 HAL\_PCD\_EP\_GetRxCount

Function Name	<code>uint16_t HAL_PCD_EP_GetRxCount ( <i>PCD_HandleTypeDef</i> * hpcd, uint8_t ep_addr)</code>
Function Description	Get Received Data Size.
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	<ul style="list-style-type: none"> <li>• <b>Data Size</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 26.2.31 HAL\_PCD\_EP\_Transmit

Function Name	<code>HAL_StatusTypeDef HAL_PCD_EP_Transmit (</code> <i>PCD_HandleTypeDef</i> * <code>hpcd, uint8_t ep_addr, uint8_t * pBuf,</code> <code>uint32_t len)</code>
Function Description	Send an amount of data.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> <li>• <b>ep_addr</b> : endpoint address</li> <li>• <b>pBuf</b> : pointer to the transmission buffer</li> <li>• <b>len</b> : amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 26.2.32 HAL\_PCD\_EP\_SetStall

Function Name	<code>HAL_StatusTypeDef HAL_PCD_EP_SetStall (</code> <i>PCD_HandleTypeDef</i> * <code>hpcd, uint8_t ep_addr)</code>
Function Description	Set a STALL condition over 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	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.33 HAL\_PCD\_EP\_ClrStall

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_ClrStall (</b> <b>PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
Function Description	Clear a STALL condition over in 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	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.34 HAL\_PCD\_EP\_Flush

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Flush (</b> <b>PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
Function Description	Flush 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	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 26.2.35 HAL\_PCD\_ActiveRemoteWakeup

Function Name	<b>HAL_StatusTypeDef HAL_PCD_ActiveRemoteWakeup (</b>
---------------	-------------------------------------------------------

***PCD\_HandleTypeDef \* hpcd)***

Function Description	HAL_PCD_ActiveRemoteWakeup : active remote wakeup signalling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**26.2.36 HAL\_PCD\_DeActiveRemoteWakeup**

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DeActiveRemoteWakeup (</b> <b><i>PCD_HandleTypeDef * hpcd)</i></b>
Function Description	HAL_PCD_DeActiveRemoteWakeup : de-active remote wakeup signalling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**26.2.37 HAL\_PCD\_GetState**

Function Name	<b>PCD_StateTypeDef HAL_PCD_GetState (</b> <b><i>PCD_HandleTypeDef * hpcd)</i></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>• <b>HAL state</b></li> </ul>

Notes                    • None.

## 26.3 PCD Firmware driver defines

### 26.3.1 PCD

PCD

***PCD Core PHY***

PCD\_PHY\_EMBEDDED

***PCD Core Speed***

PCD\_SPEED\_HIGH

PCD\_SPEED\_FULL

***PCD\_ENDP\_Type***

PCD\_ENDP0

PCD\_ENDP1

PCD\_ENDP2

PCD\_ENDP3

PCD\_ENDP4

PCD\_ENDP5

PCD\_ENDP6

PCD\_ENDP7

PCD\_SNG\_BUF

PCD\_DBL\_BUF

IS\_PCD\_ALL\_INSTANCE

***PCD EP0 MPS***

DEP0CTL\_MPS\_64

DEP0CTL\_MPS\_32

DEP0CTL\_MPS\_16

DEP0CTL\_MPS\_8

PCD\_EP0MPS\_64

PCD\_EP0MPS\_32

PCD\_EP0MPS\_16

PCD\_EP0MPS\_08

***PCD EP Type***

PCD\_EP\_TYPE\_CTRL

PCD\_EP\_TYPE\_ISOC

PCD\_EP\_TYPE\_BULK

PCD\_EP\_TYPE\_INTR

***PCD Exported Macros***

\_HAL\_PCD\_GET\_FLAG

<code>_HAL_PCD_CLEAR_FLAG</code>	
<code>USB EXTI_LINE_WAKEUP</code>	External interrupt line 18 Connected to the USB FS EXTI Line
<code>_HAL_USB_EXTI_ENABLE_IT</code>	
<code>_HAL_USB_EXTI_DISABLE_IT</code>	
<code>_HAL_USB_EXTI_GENERATE_SWIT</code>	
<b><i>PCD Private Define</i></b>	
<code>BTABLE_ADDRESS</code>	
<b><i>PCD Private Macros</i></b>	
<code>PCD_SET_ENDPOINT</code>	
<code>PCD_GET_ENDPOINT</code>	
<code>PCD_SET_EPTYPE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>sets the type in the endpoint register(bits EP_TYPE[1:0])</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>USBx: USB peripheral instance register address.</li> <li>bEpNum: Endpoint Number.</li> <li>wType: Endpoint Type.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None:</li> </ul>
<code>PCD_GET_EPTYPE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>gets the type in the endpoint register(bits EP_TYPE[1:0])</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>USBx: USB peripheral instance register address.</li> <li>bEpNum: Endpoint Number.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>Endpoint: Type</li> </ul>
<code>PCD_FreeUserBuffer</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>free buffer used from the application realizing it to the line toggles bit SW_BUF in the double buffered endpoint register</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>USBx: USB peripheral instance register address.</li> <li>bEpNum: Endpoint Number.</li> <li>bDir: Direction</li> </ul> <p><b>Return value:</b></p>

**PCD\_GET\_DB\_DIR**

- 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\_RX[1:0] & STAT\_TX[1:0])

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wStaterx: new state.
- wStatetx: new state.

**Return value:**

- None:

PCD_GET_EP_TX_STATUS	<b>Description:</b> <ul style="list-style-type: none"><li>gets the status for tx/rx transfer (bits STAT_TX[1:0] /STAT_RX[1:0])</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>USBx: USB peripheral instance register address.</li><li>bEpNum: Endpoint Number.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>status:</li></ul>
PCD_GET_EP_RX_STATUS	
PCD_SET_EP_TX_VALID	<b>Description:</b> <ul style="list-style-type: none"><li>sets directly the VALID tx/rx-status into the endpoint register</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>USBx: USB peripheral instance register address.</li><li>bEpNum: Endpoint Number.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None:</li></ul>
PCD_SET_EP_RX_VALID	
PCD_GET_EP_TX_STALL_STATUS	<b>Description:</b> <ul style="list-style-type: none"><li>checks stall condition in an endpoint.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>USBx: USB peripheral instance register address.</li><li>bEpNum: Endpoint Number.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>TRUE: = endpoint in stall condition.</li></ul>
PCD_GET_EP_RX_STALL_STATUS	
PCD_SET_EP_KIND	<b>Description:</b> <ul style="list-style-type: none"><li>set &amp; clear EP_KIND bit.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>USBx: USB peripheral instance register address.</li><li>bEpNum: Endpoint Number.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None:</li></ul>
PCD_CLEAR_EP_KIND	
PCD_SET_OUT_STATUS	<b>Description:</b>

- 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

**Description:**

- Gets address in an endpoint register.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- None:

PCD\_EP\_TX\_ADDRESS

PCD\_EP\_TX\_CNT

PCD\_EP\_RX\_ADDRESS

PCD\_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.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

**Return value:**

- address: of the buffer.

PCD\_GET\_EP\_RX\_ADDRESS

PCD\_CALC\_BLK32

**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\_SET\_EP\_RX\_CNT

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.

**Parameters:**

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.

**Return value:**

- Counter: value

PCD\_SET\_EP\_DBUF1\_ADDR

PCD\_SET\_EP\_DBUF\_ADDR

**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

## 27 HAL PCD Extension Driver

### 27.1 PCDEEx Firmware driver API description

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

#### 27.1.1 Peripheral extended features methods

- [\*HAL\\_PCDEEx\\_PMAConfig\(\)\*](#)

#### 27.1.2 **HAL\_PCDEEx\_PMAConfig**

Function Name	<b>HAL_StatusTypeDef HAL_PCDEEx_PMAConfig (</b> <i>PCD_HandleTypeDef</i> * <b>hpcd</b> , <b>uint16_t ep_addr</b> , <b>uint16_t ep_kind</b> , <b>uint32_t pmaaddress</b> )
Function Description	Configure PMA for EP.
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_kind</b> : endpoint Kind<ul style="list-style-type: none"><li>- <b>USB_SNG_BUF</b> Single Buffer used</li><li>- <b>USB_DBL_BUF</b> Double Buffer used</li></ul></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>• : <b>status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 28 HAL PWR Generic Driver

### 28.1 PWR Firmware driver API description

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

#### 28.1.1 Initialization and de-initialization functions

After reset, the backup domain (RTC registers, RTC backup data registers) is protected against possible unwanted write accesses. To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the \_\_PWR\_CLK\_ENABLE() macro.
- Enable access to RTC domain using the HAL\_PWR\_EnableBkUpAccess() function.

#### 28.1.2 Peripheral Control functions

##### WakeUp pin configuration

- WakeUp pin is used to wakeup the system from Standby mode. This pin is forced in input pull down configuration and is active on rising edges.
- There are two WakeUp pins, and up to eight Wakeup pins on STM32F07x & STM32F09x devices.
  - WakeUp Pin 1 on PA.00.
  - WakeUp Pin 2 on PC.13.
  - WakeUp Pin 3 on PE.06.(STM32F07x/STM32F09x)
  - WakeUp Pin 4 on PA.02.(STM32F07x/STM32F09x)
  - WakeUp Pin 5 on PC.05.(STM32F07x/STM32F09x)
  - WakeUp Pin 6 on PB.05.(STM32F07x/STM32F09x)
  - WakeUp Pin 7 on PB.15.(STM32F07x/STM32F09x)
  - WakeUp Pin 8 on PF.02.(STM32F07x/STM32F09x)

##### Low Power modes configuration

The devices feature 3 low-power modes:

- Sleep mode: Cortex-M0 core stopped, peripherals kept running.
- Stop mode: all clocks are stopped, regulator running, regulator in low power mode
- Standby mode: 1.2V domain powered off (mode not available on STM32F0x8 devices).

##### Sleep mode

- Entry: The Sleep mode is entered by using the HAL\_PWR\_EnterSLEEPMode(PWR\_MAINREGULATOR\_ON, PWR\_SLEEPENTRY\_WFx) functions with
  - PWR\_SLEEPENTRY\_WFI: enter SLEEP mode with WFI instruction
  - PWR\_SLEEPENTRY\_WFE: enter SLEEP mode with WFE instruction

- Exit:
  - Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.

### Stop mode

In Stop mode, all clocks in the 1.8V domain are stopped, the PLL, the HSI, and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. The voltage regulator can be configured either in normal or low-power mode. To minimize the consumption.

- Entry: The Stop mode is entered using the HAL\_PWR\_EnterSTOPMode(PWR\_MAINREGULATOR\_ON, PWR\_STOPENTRY\_WFI ) function with:
  - Main regulator ON.
  - Low Power regulator ON.
  - PWR\_STOPENTRY\_WFI: enter STOP mode with WFI instruction
  - PWR\_STOPENTRY\_WFE: enter STOP mode with WFE instruction
- Exit:
  - Any EXTI Line (Internal or External) configured in Interrupt/Event mode.
  - Some specific communication peripherals (CEC, USART, I2C) interrupts, when programmed in wakeup mode (the peripheral must be programmed in wakeup mode and the corresponding interrupt vector must be enabled in the NVIC)

### Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M0 deep sleep mode, with the voltage regulator disabled. The 1.8V domain is consequently powered off. The PLL, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for the RTC registers, RTC backup registers and Standby circuitry. The voltage regulator is OFF.

- Entry:
  - The Standby mode is entered using the HAL\_PWR\_EnterSTANDBYMode() function.
- Exit:
  - WKUP pin rising edge, RTC alarm (Alarm A), RTC wakeup, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

### Auto-wakeup (AWU) from low-power mode

The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event, a time-stamp event, or a comparator event, without depending on an external interrupt (Auto-wakeup mode).

- RTC auto-wakeup (AWU) from the Stop and Standby modes
  - To wake up from the Stop mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL\_RTC\_SetAlarm\_IT() function.
  - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to configure the RTC to detect the tamper or time stamp event using the HAL\_RTC\_SetTimeStamp\_IT() or HAL\_RTC\_SetTamper\_IT() functions.
  - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL\_RTC\_SetWakeUpTimer\_IT() function.
- Comparator auto-wakeup (AWU) from the Stop mode

- To wake up from the Stop mode with a comparator wakeup event, it is necessary to:
  - Configure the EXTI Line associated with the comparator (example EXTI Line 22 for comparator 2) to be sensitive to the selected edges (falling, rising or falling and rising) (Interrupt or Event modes) using the EXTI\_Init() function.
  - Configure the comparator to generate the event.
- ***HAL\_PWR\_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\_EnableBkUpAccess()***
- ***HAL\_PWR\_DisableBkUpAccess()***

### 28.1.3 HAL\_PWR\_EnableBkUpAccess

Function Name	<b>void HAL_PWR_EnableBkUpAccess ( void )</b>
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers).
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.</li> </ul>

### 28.1.4 HAL\_PWR\_DisableBkUpAccess

Function Name	<b>void HAL_PWR_DisableBkUpAccess ( void )</b>
Function Description	Disables access to the backup domain (RTC registers, RTC backup data registers).
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.</li> </ul>

## 28.1.5 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 value of : PWREx Wakeup Pins</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 28.1.6 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 values of : PWREx Wakeup Pins</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 28.1.7 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> : Specifies the regulator state in SLEEP mode. On STM32F0 devices, this parameter is a dummy value and it is ignored as regulator can't be modified in this mode. Parameter is kept for platform compatibility.</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: <ul style="list-style-type: none"> <li>– <b>PWR_SLEEPENTRY_WFI</b> enter SLEEP mode with WFI instruction</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- <b>PWR_SLEEPENTRY_WFE</b> 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>

### 28.1.8 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: <ul style="list-style-type: none"> <li>- <b>PWR_MAINREGULATOR_ON</b> STOP mode with regulator ON</li> <li>- <b>PWR_LOWPOWERREGULATOR_ON</b> STOP mode with low power regulator ON</li> </ul> </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: <ul style="list-style-type: none"> <li>- <b>PWR_STOPENTRY_WFI</b> Enter STOP mode with WFI instruction</li> <li>- <b>PWR_STOPENTRY_WFE</b> Enter STOP mode with WFE instruction</li> </ul> </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 issuing an interrupt or a wakeup event, the 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>

### 28.1.9 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)RTC alternate function pins if configured for tamper, time-stamp, RTC Alarm out, or RTC clock calibration out.WKUP pins if enabled. STM32F0x8 devices, the Stop mode is available, but it is meaningless to distinguish between voltage regulator in Low power mode and voltage regulator in Run mode because the regulator not used and the core is supplied directly from an external source. Consequently, the Standby mode is not available on those devices.</li> </ul>

### 28.1.10 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>

### 28.1.11 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>

### 28.1.12 HAL\_PWR\_EnableSEVOnPend

Function Name	<b>void HAL_PWR_EnableSEVOnPend ( void )</b>
Function Description	Enables CORTEX M4 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>

### 28.1.13 HAL\_PWR\_DisableSEVOnPend

Function Name	<b>void HAL_PWR_DisableSEVOnPend ( void )</b>
Function Description	Disables CORTEX M4 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>

### 28.1.14 HAL\_PWR\_EnableBkUpAccess

Function Name	<b>void HAL_PWR_EnableBkUpAccess ( void )</b>
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers).
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.</li></ul>

### 28.1.15 HAL\_PWR\_DisableBkUpAccess

Function Name	<b>void HAL_PWR_DisableBkUpAccess ( void )</b>
Function Description	Disables access to the backup domain (RTC registers, RTC backup data registers).
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.</li> </ul>

## 28.2 PWR Firmware driver defines

### 28.2.1 PWR

PWR

*PWR Exported Macro*

`_HAL_PWR_GET_FLAG`

**Description:**

- Check PWR flag is set or not.

**Parameters:**

- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
  - `PWR_FLAG_WU`: Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm (Alarm A), RTC Tamper event, RTC TimeStamp event or RTC Wakeup. An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.
  - `PWR_FLAG_SB`: StandBy flag. This flag indicates that the system was resumed from StandBy mode.
  - `PWR_FLAG_PVDO`: PVD Output. This flag is valid only if PVD is enabled by the `HAL_PWR_EnablePVD()` function. The PVD is stopped by Standby mode For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set. Warning: this Flag is not available on STM32F030x8 products
  - `PWR_FLAG_VREFINTRDY`: This flag indicates that the internal reference voltage VREFINT is ready. Warning: this Flag is not available on

**Return value:**

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

[\\_\\_HAL\\_PWR\\_CLEAR\\_FLAG](#)**Description:**

- Clear the PWR's pending flags.

**Parameters:**

- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be one of the following values:
  - PWR\_FLAG\_WU: Wake Up flag
  - PWR\_FLAG\_SB: StandBy flag

**PWR Regulator state in STOP mode**

PWR\_MAINREGULATOR\_ON

PWR\_LOWPOWERREGULATOR\_ON

IS\_PWR\_REGULATOR

**PWR SLEEP mode entry**

PWR\_SLEEPENTRY\_WFI

PWR\_SLEEPENTRY\_WFE

IS\_PWR\_SLEEP\_ENTRY

**PWR STOP mode entry**

PWR\_STOPENTRY\_WFI

PWR\_STOPENTRY\_WFE

IS\_PWR\_STOP\_ENTRY

## 29 HAL PWR Extension Driver

### 29.1 PWREx Firmware driver registers structures

#### 29.1.1 PWR\_PVDTTypeDef

*PWR\_PVDTTypeDef* is defined in the `stm32f0xx_hal_pwr_ex.h`

##### Data Fields

- *uint32\_t PVDLevel*
- *uint32\_t Mode*

##### Field Documentation

- *uint32\_t PWR\_PVDTTypeDef::PVDLevel* PVDLevel: Specifies the PVD detection level This parameter can be a value of [PWREx\\_PVD\\_detection\\_level](#)
- *uint32\_t PWR\_PVDTTypeDef::Mode* Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [PWREx\\_PVD\\_Mode](#)

### 29.2 PWREx Firmware driver API description

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

#### 29.2.1 Peripheral extended control functions

##### PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in the PWR\_CR).
- A PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through `HAL_PWR_PVDConfig()`, `HAL_PWR_EnablePVD()` functions.
- The PVD is stopped in Standby mode. PVD is not available on STM32F030x4/x6/x8

##### VDDIO2 Monitor Configuration

- VDDIO2 monitor is used to monitor the VDDIO2 power supply by comparing it to VREFInt Voltage
- This monitor is internally connected to the EXTI line31 and can generate an interrupt if enabled. This is done through `HAL_PWR_EnableVddio2Monitor()` function. VDDIO2 is available on STM32F07x/09x/04x
- [`HAL\_PWR\_PVDConfig\(\)`](#)
- [`HAL\_PWR\_EnablePVD\(\)`](#)
- [`HAL\_PWR\_DisablePVD\(\)`](#)
- [`HAL\_PWR\_PVD\_IRQHandler\(\)`](#)

- [\*\*\*HAL\\_PWR\\_PVDCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_PWR\\_EnableVddio2Monitor\(\)\*\*\*](#)
- [\*\*\*HAL\\_PWR\\_DisableVddio2Monitor\(\)\*\*\*](#)
- [\*\*\*HAL\\_PWR\\_Vddio2Monitor\\_IRQHandler\(\)\*\*\*](#)
- [\*\*\*HAL\\_PWR\\_Vddio2MonitorCallback\(\)\*\*\*](#)

## 29.2.2 HAL\_PWR\_PVDConfig

Function Name	<b>void HAL_PWR_PVDConfig ( <a href="#"><i>PWR_PVDTTypeDef</i></a> * <b>sConfigPVD</b> )</b>
Function Description	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"> <li>• <b>sConfigPVD</b> : pointer to an <i>PWR_PVDTTypeDef</i> structure that contains the configuration information for the PVD.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.</li> </ul>

## 29.2.3 HAL\_PWR\_EnablePVD

Function Name	<b>void HAL_PWR_EnablePVD ( void )</b>
Function Description	Enables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 29.2.4 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.5 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() or PVD_VDDIO2_IRQHandler().</li></ul>

### 29.2.6 HAL\_PWR\_PVDCallback

Function Name	<b>void HAL_PWR_PVDCallback ( void )</b>
Function Description	PWR PVD interrupt callback.
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 29.2.7 HAL\_PWR\_EnableVddio2Monitor

Function Name	<b>void HAL_PWR_EnableVddio2Monitor ( void )</b>
Function Description	Enable VDDIO2 monitor: enable Exti 31 and falling edge detection.
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>If Exti 31 is enable correctly and VDDIO2 voltage goes below Vrefint, an interrupt is generated Irq line 1. NVIS has to be enable by user.</li></ul>

### 29.2.8 HAL\_PWR\_DisableVddio2Monitor

Function Name	<b>void HAL_PWR_DisableVddio2Monitor ( void )</b>
Function Description	Disable the Vddio2 Monitor.
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 29.2.9 HAL\_PWR\_Vddio2Monitor\_IRQHandler

Function Name	<b>void HAL_PWR_Vddio2Monitor_IRQHandler ( void )</b>
Function Description	This function handles the PWR Vddio2 monitor 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 VDDIO2_IRQHandler() PVD_VDDIO2_IRQHandler().</li></ul>

### 29.2.10 HAL\_PWR\_Vddio2MonitorCallback

Function Name	<b>void HAL_PWR_Vddio2MonitorCallback ( void )</b>
Function Description	PWR Vddio2 Monitor interrupt callback.
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

## 29.3 PWREx Firmware driver defines

### 29.3.1 PWREx

PWREx

*PWREx Exported Macros*

`_HAL_PWR_PVD_EXTI_ENABLE_IT`

**Description:**

- Enable interrupt on PVD Exti Line 16.

**Return value:**

- None.:

`_HAL_PWR_PVD_EXTI_DISABLE_IT`

**Description:**

- Disable interrupt on PVD Exti Line 16.

**Return value:**

- None.:

`_HAL_PWR_PVD_EXTI_ENABLE_EVENT`

**Description:**

- Enable event on PVD Exti Line 16.

**Return value:**

- None.:

`_HAL_PWR_PVD_EXTI_DISABLE_EVENT`

**Description:**

- Disable event on PVD Exti Line 16.

**Return value:**

- None.:

`_HAL_PWR_PVD_EXTI_CLEAR_EDGE_TRIGGER`

**Description:**

- PVD EXTI line configuration: clear falling edge and rising edge trigger.

**Return value:**

- None.:

`_HAL_PWR_PVD_EXTI_SET_FALLING_EDGE_TRIGGER`

**Description:**

- PVD EXTI line configuration: set falling edge trigger.

**Return value:**

- None.:

`_HAL_PWR_PVD_EXTI_SET_RISING_EDGE_TRIGGER`

**Description:**

- PVD EXTI line configuration: set rising edge trigger.

**Return value:**

`_HAL_PWR_PVD_EXTI_GET_FLAG`

- None.:

**Description:**

- Check whether the specified PVD EXTI interrupt flag is set or not.

**Return value:**

- EXTI: PVD Line Status.

`_HAL_PWR_PVD_EXTI_CLEAR_FLAG`

**Description:**

- Clear the PVD EXTI flag.

**Return value:**

- None.:

`_HAL_PWR_PVD_EXTI_GENERATE_SWIT`

**Description:**

- Generate a Software interrupt on selected EXTI line.

**Return value:**

- None.:

`_HAL_PWR_VDDIO2_EXTI_ENABLE_IT`

**Description:**

- Enable interrupt on Vddio2 Monitor Exti Line 31.

**Return value:**

- None.:

`_HAL_PWR_VDDIO2_EXTI_DISABLE_IT`

**Description:**

- Disable interrupt on Vddio2 Monitor Exti Line 31.

**Return value:**

- None.:

`_HAL_PWR_VDDIO2_EXTI_CLEAR_EDGE_TRIGGER`

**Description:**

- Vddio2 Monitor EXTI line configuration: clear falling edge and rising edge trigger.

**Return value:**

- None.:

<code>__HAL_PWR_VDDIO2_EXTI_SET_FALLING_EDGE_TRIGGER</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Vddio2 Monitor EXTI line configuration: set falling edge trigger.</li></ul>
	<b>Return value:</b> <ul style="list-style-type: none"><li>• None.:</li></ul>
<code>__HAL_PWR_VDDIO2_EXTI_GET_FLAG</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Check whether the specified VDDIO2 monitor EXTI interrupt flag is set or not.</li></ul>
	<b>Return value:</b> <ul style="list-style-type: none"><li>• EXTI: VDDIO2 Monitor Line Status.</li></ul>
<code>__HAL_PWR_VDDIO2_EXTI_CLEAR_FLAG</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Clear the VDDIO2 Monitor EXTI flag.</li></ul>
	<b>Return value:</b> <ul style="list-style-type: none"><li>• None.:</li></ul>
<code>__HAL_PWR_VDDIO2_EXTI_GENERATE_SWIT</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Generate a Software interrupt on selected EXTI line.</li></ul>
	<b>Return value:</b> <ul style="list-style-type: none"><li>• None.:</li></ul>

**PWREx EXTI Line**

<code>PWR_EXTI_LINE_PVD</code>	External interrupt line 16 Connected to the PVD EXTI Line
<code>PWR_EXTI_LINE_VDDIO2</code>	External interrupt line 31 Connected to the Vddio2 Monitor EXTI Line

**PWREx Flag**

<code>PWR_FLAG_WU</code>	
<code>PWR_FLAG_SB</code>	
<code>PWR_FLAG_PVDO</code>	
<code>PWR_FLAG_VREFINTRDY</code>	

**PWREx Private Constants**

<code>PVD_MODE_IT</code>
--------------------------

PVD_MODE_EVT	
PVD_RISING_EDGE	
PVD_FALLING_EDGE	
<b>PWREx PVD detection level</b>	
PWR_PVDLEVEL_0	
PWR_PVDLEVEL_1	
PWR_PVDLEVEL_2	
PWR_PVDLEVEL_3	
PWR_PVDLEVEL_4	
PWR_PVDLEVEL_5	
PWR_PVDLEVEL_6	
PWR_PVDLEVEL_7	
IS_PWR_PVD_LEVEL	
<b>PWREx PVD Mode</b>	
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
IS_PWR_PVD_MODE	
<b>PWREx Wakeup Pins</b>	
PWR_WAKEUP_PIN1	
PWR_WAKEUP_PIN2	
PWR_WAKEUP_PIN3	
PWR_WAKEUP_PIN4	
PWR_WAKEUP_PIN5	
PWR_WAKEUP_PIN6	
PWR_WAKEUP_PIN7	
PWR_WAKEUP_PIN8	
IS_PWR_WAKEUP_PIN	

## 30 HAL RCC Generic Driver

### 30.1 RCC Firmware driver registers structures

#### 30.1.1 RCC\_PLLInitTypeDef

*RCC\_PLLInitTypeDef* is defined in the `stm32f0xx_hal_rcc.h`

##### Data Fields

- *uint32\_t PLLState*
- *uint32\_t PLLSource*
- *uint32\_t PREDIV*
- *uint32\_t PLLMUL*

##### Field Documentation

- *uint32\_t RCC\_PLLInitTypeDef::PLLState* 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::PREDIV* PREDIV: Predivision factor for PLL VCO input clock This parameter must be a value of [RCC\\_PLL\\_Prediv\\_Factor](#)
- *uint32\_t RCC\_PLLInitTypeDef::PLLMUL* PLLMUL: Multiplication factor for PLL VCO input clock This parameter must be a value of [RCC\\_PLL\\_Multiplication\\_Factor](#)

#### 30.1.2 RCC\_OscInitTypeDef

*RCC\_OscInitTypeDef* is defined in the `stm32f0xx_hal_rcc.h`

##### Data Fields

- *uint32\_t OscillatorType*
- *uint32\_t HSEState*
- *uint32\_t LSEState*
- *uint32\_t HSISState*
- *uint32\_t HSICalibrationValue*
- *uint32\_t HSI14State*
- *uint32\_t HSI14CalibrationValue*
- *uint32\_t HSI48State*
- *uint32\_t LSISState*
- *RCC\_PLLInitTypeDef PLL*

##### Field Documentation

- *uint32\_t RCC\_OscInitTypeDef::OscillatorType* The Oscillators to be configured. This parameter can be a value of [RCC\\_Oscillator\\_Type](#)
- *uint32\_t RCC\_OscInitTypeDef::HSEState* The new state of the HSE. This parameter can be a value of [RCC\\_HSE\\_Config](#)
- *uint32\_t RCC\_OscInitTypeDef::LSEState* The new state of the LSE. This parameter can be a value of [RCC\\_LSE\\_Config](#)

- ***uint32\_t RCC\_OsclInitTypeDef::HSIState*** The new state of the HSI. This parameter can be a value of [RCC\\_HSI\\_Config](#)
- ***uint32\_t RCC\_OsclInitTypeDef::HSICalibrationValue*** The HSI calibration trimming value (default is RCC\_HSICALIBRATION\_DEFAULT). This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F
- ***uint32\_t RCC\_OsclInitTypeDef::HSI14State*** The new state of the HSI14. This parameter can be a value of [RCC\\_HSI14\\_Config](#)
- ***uint32\_t RCC\_OsclInitTypeDef::HSI14CalibrationValue*** The HSI14 calibration trimming value (default is RCC\_HSI14CALIBRATION\_DEFAULT). This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F
- ***uint32\_t RCC\_OsclInitTypeDef::HSI48State*** The new state of the HSI48 (only applicable to STM32F07x, STM32F0x2 and STM32F09x devices). This parameter can be a value of [RCCEEx\\_HSI48\\_Config](#)
- ***uint32\_t RCC\_OsclInitTypeDef::LSIState*** The new state of the LSI. This parameter can be a value of [RCC\\_LSI\\_Config](#)
- ***RCC\_PLLInitTypeDef RCC\_OsclInitTypeDef::PLL*** PLL structure parameters

### 30.1.3 RCC\_ClkInitTypeDef

*RCC\_ClkInitTypeDef* is defined in the `stm32f0xx_hal_rcc.h`

#### Data Fields

- ***uint32\_t ClockType***
- ***uint32\_t SYSSCLKSource***
- ***uint32\_t AHBCLKDivider***
- ***uint32\_t APB1CLKDivider***

#### 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::SYSSCLKSource*** The clock source (SYSSCLK) 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 (SYSSCLK). 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\\_Clock\\_Source](#)

## 30.2 RCC Firmware driver API description

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

### 30.2.1 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 disabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHB) and Low speed (APB) busses; all peripherals mapped on these busses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in input floating state, except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB busses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals which clocks are not derived from the System clock (RTC, ADC, I2C, USART, TIM, USB FS, etc..)

### 30.2.2 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 registeres.

- This delay depends on the peripheral mapping.
- If peripheral is mapped on AHB: the delay is 2 AHB clock cycle after the clock enable bit is set on the hardware register
- If peripheral is mapped on APB: the delay is 2 APB clock cycle after the clock enable bit is set on the hardware register

Possible Workarounds:

1. Enable the peripheral clock sometimes before the peripheral read/write register is required.
2. For AHB peripheral, insert two dummy read to the peripheral register.
3. For APB peripheral, insert a dummy read to the peripheral register.

### 30.2.3 Initialization and de-initialization function

This section provide functions allowing to configure the internal/external oscillators (HSE, HSI, HSI14, HSI48, LSE, LSI, PLL, CSS and MCO) and the System busses clocks (SYSCLK, AHB and APB1).

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. The HSI clock can be used also to clock the USART and I2C peripherals.
2. HSI14 (high-speed internal), 14 MHz factory-trimmed RC used directly to clock the ADC peripheral.
3. LSI (low-speed internal), 40 KHz low consumption RC used as IWDG and/or RTC clock source.
4. HSE (high-speed external), 4 to 32 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
5. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
6. PLL (clocked by HSI, HSI48 or HSE), featuring different output clocks:
  - The first output is used to generate the high speed system clock (up to 48 MHz)
  - The second output is used to generate the clock for the USB FS (48 MHz)

- The third output may be used to generate the clock for the TIM, I2C and USART peripherals (up to 48 MHz)
- 7. 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-M0 NMI (Non-Maskable Interrupt) exception vector.
- 8. MCO (microcontroller clock output), used to output SYSCLK, HSI, HSE, LSI, LSE or PLL clock (divided by 2) output on pin (such as PA8 pin).

System, AHB and APB busses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): HSI, HSE and PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...). APB1 (PCLK1) clock is derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "`HAL_RCC_GetSysClockFreq()`" function to retrieve the frequencies of these clocks.
2. All the peripheral clocks are derived from the System clock (SYSCLK) except:
  - The FLASH program/erase clock which is always HSI 8MHz clock.
  - The USB 48 MHz clock which is derived from the PLL VCO clock.
  - The USART clock which can be derived as well from HSI 8MHz, LSI or LSE.
  - The I2C clock which can be derived as well from HSI 8MHz clock.
  - The ADC clock which is derived from PLL output.
  - The RTC clock which is derived from the LSE, LSI or 1 MHz HSE\_RTC (HSE divided by a programmable prescaler). The System clock (SYSCLK) frequency must be higher or equal to the RTC clock frequency.
  - IWDG clock which is always the LSI clock.
3. For the STM32F0xx devices, the maximum frequency of the SYSCLK, HCLK and PCLK1 is 48 MHz, Depending on the SYSCLK frequency, the flash latency should be adapted according to the below table.
4. After reset, the System clock source is the HSI (8 MHz) with 0 WS and prefetch is disabled.

**Table 20: Number of wait states (WS) according to system clock (SYSCLK) frequency**

Latency	SYSCLK clock frequency (MHz)
0WS (1CPU cycle)	$0 \leq \text{SYSCLK} \leq 24$
1WS (2CPU cycles)	$24 < \text{HCLK} \leq 48$

- `HAL_RCC_DelInit()`
- `HAL_RCC_OscConfig()`
- `HAL_RCC_ClockConfig()`

### 30.2.4 Peripheral Control function

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_GetOscConfig()`
- `HAL_RCC_GetClockConfig()`
- `HAL_RCC_NMI_IRQHandler()`
- `HAL_RCC_CCSCallback()`

### 30.2.5 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 prescaler set to 1.CSS, MCO OFFAll interrupts disabled</li> <li>• This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks</li> </ul>

### 30.2.6 HAL\_RCC\_OscConfig

Function Name	<b>HAL_StatusTypeDef HAL_RCC_OscConfig (</b> <b>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>• <b>HAL status</b></li> </ul>

Notes

- The PLL is not disabled when used as system clock.

### 30.2.7 HAL\_RCC\_ClockConfig

Function Name	<b>HAL_StatusTypeDef HAL_RCC_ClockConfig (</b> <b>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 <b>RCC_ClkInitStruct</b> .
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_ClkInitStruct</b> : pointer to an <b>RCC_ClkInitTypeDef</b> 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: <ul style="list-style-type: none"> <li>– <b>FLASH_LATENCY_0</b> FLASH 0 Latency cycle</li> <li>– <b>FLASH_LATENCY_1</b> FLASH 1 Latency cycle</li> </ul> </li> </ul>
Return values	• <b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>• The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by <b>HAL_RCC_GetHCLKFreq()</b> 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.</li> </ul>

### 30.2.8 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(such as PA8).
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: <ul style="list-style-type: none"> <li>– <b>RCC_MCO</b> Clock source to output on MCO pin(such as PA8).</li> </ul> </li> <li>• <b>RCC_MCOSource</b> : specifies the clock source to output. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>RCC_MCOSOURCE_LSI</b> LSI clock selected as MCO source</li> <li>– <b>RCC_MCOSOURCE_HSI</b> HSI clock selected as MCO source</li> <li>– <b>RCC_MCOSOURCE_LSE</b> LSE clock selected as MCO source</li> <li>– <b>RCC_MCOSOURCE_HSE</b> HSE clock selected as MCO source</li> <li>– <b>RCC_MCOSOURCE_PLLCLK_NODIV</b> main PLL clock not divided selected as MCO source (not applicable to</li> </ul> </li> </ul>

---

	STM32F05x devices)
	– <b>RCC_MCOSOURCE_PLLCLK_DIV2</b> main PLL clock divided by 2 selected as MCO source
	– <b>RCC_MCOSOURCE_SYSCLK</b> System clock (SYSCLK) selected as MCO source
•	<b>RCC_MCODiv</b> : specifies the MCOx prescaler. This parameter can be one of the following values:
	– <b>RCC_MCO_NODIV</b> no division applied to MCO clock
Return values	• None.
Notes	• MCO pin (such as PA8) should be configured in alternate function mode.

### 30.2.9 HAL\_RCC\_EnableCSS

Function Name	<b>void HAL_RCC_EnableCSS ( void )</b>
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-M0 NMI (Non-Maskable Interrupt) exception vector.

### 30.2.10 HAL\_RCC\_DisableCSS

Function Name	<b>void HAL_RCC_DisableCSS ( void )</b>
Function Description	Disables the Clock Security System.
Return values	• None.
Notes	• None.

### 30.2.11 HAL\_RCC\_GetSysClockFreq

Function Name	<code>uint32_t HAL_RCC_GetSysClockFreq ( void )</code>
Function Description	Returns the SYSCLK frequency.
Return values	<ul style="list-style-type: none"> <li>• <b>SYSCLK frequency</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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:</li> <li>• If SYSCLK source is HSI, function returns a value based on HSI_VALUE(*)</li> <li>• If SYSCLK source is HSI48, function returns a value based on HSI48_VALUE(*)</li> <li>• If SYSCLK source is HSE, function returns a value based on HSE_VALUE divided by PREDIV factor(**)</li> <li>• If SYSCLK source is PLL, function returns a value based on HSE_VALUE divided by PREDIV factor(**) or depending on STM32F0xx devices either a value based on HSI_VALUE divided by 2 or HSI_VALUE divided by PREDIV factor(*) multiplied by the PLL factor .</li> <li>• (*) HSI_VALUE &amp; HSI48_VALUE are constants defined in stm32f0xx_hal_conf.h file (default values 8 MHz and 48MHz).</li> <li>• (**) HSE_VALUE is a constant defined in stm32f0xx_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.</li> <li>• The result of this function could be not correct when using fractional value for HSE crystal.</li> <li>• This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.</li> <li>• 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.</li> </ul>

### 30.2.12 HAL\_RCC\_GetHCLKFreq

Function Name	<code>uint32_t HAL_RCC_GetHCLKFreq ( void )</code>
Function Description	Returns the HCLK frequency.
Return values	<ul style="list-style-type: none"> <li>• <b>HCLK frequency</b></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> </ul>

- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function

### 30.2.13 HAL\_RCC\_GetPCLK1Freq

Function Name	<code>uint32_t HAL_RCC_GetPCLK1Freq ( void )</code>
Function Description	Returns the PCLK1 frequency.
Return values	<ul style="list-style-type: none"> <li><b>PCLK1 frequency</b></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>

### 30.2.14 HAL\_RCC\_GetOscConfig

Function Name	<code>void HAL_RCC_GetOscConfig ( <i>RCC_OscInitTypeDef</i> * RCC_OscInitStruct)</code>
Function Description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li><b>RCC_OscInitStruct</b> : pointer to an RCC_OscInitTypeDef structure that will be configured.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 30.2.15 HAL\_RCC\_GetClockConfig

Function Name	<code>void HAL_RCC_GetClockConfig ( <i>RCC_ClkInitTypeDef</i> * RCC_ClkInitStruct, uint32_t * pFLatency)</code>
Function Description	Get the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters	<ul style="list-style-type: none"><li>• <b>RCC_ClkInitStruct</b> : pointer to an RCC_ClkInitTypeDef structure that contains the current clock configuration.</li><li>• <b>pFLatency</b> : Pointer on the Flash Latency.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 30.2.16 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 NMI_Handler().</li></ul>

### 30.2.17 HAL\_RCC\_CCSCallback

Function Name	<b>void HAL_RCC_CCSCallback ( void )</b>
Function Description	RCC Clock Security System interrupt callback.
Return values	<ul style="list-style-type: none"><li>• <b>none</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 30.3 RCC Firmware driver defines

### 30.3.1 RCC

RCC

**RCC AHB Clock Enable Disable**

`_GPIOA_CLK_ENABLE`  
`_GPIOB_CLK_ENABLE`  
`_GPIOC_CLK_ENABLE`

\_\_GPIOF\_CLK\_ENABLE  
\_\_CRC\_CLK\_ENABLE  
\_\_DMA1\_CLK\_ENABLE  
\_\_SRAM\_CLK\_ENABLE  
\_\_FLITF\_CLK\_ENABLE  
\_\_GPIOA\_CLK\_DISABLE  
\_\_GPIOB\_CLK\_DISABLE  
\_\_GPIOC\_CLK\_DISABLE  
\_\_GPIOF\_CLK\_DISABLE  
\_\_CRC\_CLK\_DISABLE  
\_\_DMA1\_CLK\_DISABLE  
\_\_SRAM\_CLK\_DISABLE  
\_\_FLITF\_CLK\_DISABLE

**RCC AHB Clock Source**

RCC\_SYSCLK\_DIV1  
RCC\_SYSCLK\_DIV2  
RCC\_SYSCLK\_DIV4  
RCC\_SYSCLK\_DIV8  
RCC\_SYSCLK\_DIV16  
RCC\_SYSCLK\_DIV64  
RCC\_SYSCLK\_DIV128  
RCC\_SYSCLK\_DIV256  
RCC\_SYSCLK\_DIV512  
IS\_RCC\_SYSCLK\_DIV

**RCC AHB Force Release Reset**

\_\_AHB\_FORCE\_RESET  
\_\_GPIOA\_FORCE\_RESET  
\_\_GPIOB\_FORCE\_RESET  
\_\_GPIOC\_FORCE\_RESET  
\_\_GPIOF\_FORCE\_RESET  
\_\_AHB\_RELEASE\_RESET  
\_\_GPIOA\_RELEASE\_RESET  
\_\_GPIOB\_RELEASE\_RESET  
\_\_GPIOC\_RELEASE\_RESET  
\_\_GPIOF\_RELEASE\_RESET

**RCC APB1 Clock Enable Disable**

\_\_TIM3\_CLK\_ENABLE  
\_\_TIM14\_CLK\_ENABLE  
\_\_WWDG\_CLK\_ENABLE  
\_\_I2C1\_CLK\_ENABLE  
\_\_PWR\_CLK\_ENABLE  
\_\_TIM3\_CLK\_DISABLE  
\_\_TIM14\_CLK\_DISABLE  
\_\_WWDG\_CLK\_DISABLE  
\_\_I2C1\_CLK\_DISABLE  
\_\_PWR\_CLK\_DISABLE

**RCC APB1 Clock Source**

RCC\_HCLK\_DIV1  
RCC\_HCLK\_DIV2  
RCC\_HCLK\_DIV4  
RCC\_HCLK\_DIV8  
RCC\_HCLK\_DIV16  
IS\_RCC\_HCLK\_DIV

**RCC APB1 Force Release Reset**

\_\_APB1\_FORCE\_RESET  
\_\_TIM3\_FORCE\_RESET  
\_\_TIM14\_FORCE\_RESET  
\_\_WWDG\_FORCE\_RESET  
\_\_I2C1\_FORCE\_RESET  
\_\_PWR\_FORCE\_RESET  
\_\_APB1\_RELEASE\_RESET  
\_\_TIM3\_RELEASE\_RESET  
\_\_TIM14\_RELEASE\_RESET  
\_\_WWDG\_RELEASE\_RESET  
\_\_I2C1\_RELEASE\_RESET  
\_\_PWR\_RELEASE\_RESET

**RCC APB2 Clock Enable Disable**

\_\_SYSCFG\_CLK\_ENABLE  
\_\_ADC1\_CLK\_ENABLE  
\_\_TIM1\_CLK\_ENABLE  
\_\_SPI1\_CLK\_ENABLE  
\_\_TIM16\_CLK\_ENABLE

\_TIM17\_CLK\_ENABLE  
\_USART1\_CLK\_ENABLE  
\_DBGMCU\_CLK\_ENABLE  
\_SYSCFG\_CLK\_DISABLE  
\_ADC1\_CLK\_DISABLE  
\_TIM1\_CLK\_DISABLE  
\_SPI1\_CLK\_DISABLE  
\_TIM16\_CLK\_DISABLE  
\_TIM17\_CLK\_DISABLE  
\_USART1\_CLK\_DISABLE  
\_DBGMCU\_CLK\_DISABLE

**RCC APB2 Force Release Reset**

\_APB2\_FORCE\_RESET  
\_SYSCFG\_FORCE\_RESET  
\_ADC1\_FORCE\_RESET  
\_TIM1\_FORCE\_RESET  
\_SPI1\_FORCE\_RESET  
\_USART1\_FORCE\_RESET  
\_TIM16\_FORCE\_RESET  
\_TIM17\_FORCE\_RESET  
\_DBGMCU\_FORCE\_RESET  
\_APB2\_RELEASE\_RESET  
\_SYSCFG\_RELEASE\_RESET  
\_ADC1\_RELEASE\_RESET  
\_TIM1\_RELEASE\_RESET  
\_SPI1\_RELEASE\_RESET  
\_USART1\_RELEASE\_RESET  
\_TIM16\_RELEASE\_RESET  
\_TIM17\_RELEASE\_RESET  
\_DBGMCU\_RELEASE\_RESET

**RCC BitAddress AliasRegion**

RCC\_OFFSET  
RCC\_CR\_OFFSET  
RCC\_CFGR\_OFFSET  
RCC\_CIR\_OFFSET  
RCC\_BDCR\_OFFSET

RCC\_CSR\_OFFSET  
RCC\_CR2\_OFFSET  
RCC\_CR\_BYTE2\_ADDRESS  
RCC\_CIR\_BYTE1\_ADDRESS  
RCC\_CIR\_BYTE2\_ADDRESS  
RCC\_CSR\_BYTE1\_ADDRESS  
RCC\_BDCR\_BYTE0\_ADDRESS  
RCC\_CFGR\_PLLMUL\_BITNUMBER  
RCC\_CFGR2\_PREDIV\_BITNUMBER  
***RCC Calibration values***  
IS\_RCC\_CALIBRATION\_VALUE  
***RCC Flag***  
CR\_REG\_INDEX  
CR2\_REG\_INDEX  
BDCR\_REG\_INDEX  
CSR\_REG\_INDEX  
RCC\_CR\_HSIRDY\_BitNumber  
RCC\_CR\_HSERDY\_BitNumber  
RCC\_CR\_PLLRDY\_BitNumber  
RCC\_FLAG\_HSIRDY  
RCC\_FLAG\_HSERDY  
RCC\_FLAG\_PLLRDY  
RCC\_CR2\_HSI14RDY\_BitNumber  
RCC\_FLAG\_HSI14RDY  
RCC\_BDCR\_LSERDY\_BitNumber  
RCC\_FLAG\_LSERDY  
RCC\_CSR\_LSIRDY\_BitNumber  
RCC\_CSR\_V18PWRRSTF\_BitNumber  
RCC\_CSR\_RMVF\_BitNumber  
RCC\_CSR\_OBLRSTF\_BitNumber  
RCC\_CSR\_PINRSTF\_BitNumber  
RCC\_CSR\_PORRSTF\_BitNumber  
RCC\_CSR\_SFTRSTF\_BitNumber  
RCC\_CSR\_IWDGRSTF\_BitNumber  
RCC\_CSR\_WWDGRSTF\_BitNumber  
RCC\_CSR\_LPWRRSTF\_BitNumber

RCC\_FLAG\_LSIRDY  
 RCC\_FLAG\_V18PWRRST  
 RCC\_FLAG\_RMV  
 RCC\_FLAG\_OBLRST  
 RCC\_FLAG\_PINRST  
 RCC\_FLAG\_PORRST  
 RCC\_FLAG\_SFTRST  
 RCC\_FLAG\_IWDGRST  
 RCC\_FLAG\_WWDGRST  
 RCC\_FLAG\_LPWRRST

### ***RCC Flags Interrupts Management***

[\\_\\_HAL\\_RCC\\_ENABLE\\_IT](#)

#### **Description:**

- Enable RCC interrupt (Perform Byte access to RCC\_CIR[12:8] 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 enable
  - RCC\_IT\_LSERDY: LSE ready interrupt enable
  - RCC\_IT\_HSIRDY: HSI ready interrupt enable
  - RCC\_IT\_HSERDY: HSE ready interrupt enable
  - RCC\_IT\_PLLRDY: PLL ready interrupt enable
  - RCC\_IT\_HSI14RDY: HSI14 ready interrupt enable
  - RCC\_IT\_HSI48RDY: HSI48 ready interrupt enable (only applicable to STM32F0X2 USB devices)

[\\_\\_HAL\\_RCC\\_DISABLE\\_IT](#)

#### **Description:**

- Disable RCC interrupt (Perform Byte access to RCC\_CIR[12:8] 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

- enable
- RCC\_IT\_LSIRDY: LSE ready interrupt enable
- RCC\_IT\_HSIRDY: HSI ready interrupt enable
- RCC\_IT\_HSERDY: HSE ready interrupt enable
- RCC\_IT\_PLLRDY: PLL ready interrupt enable
- RCC\_IT\_HSI14RDY: HSI14 ready interrupt enable
- RCC\_IT\_HSI48RDY: HSI48 ready interrupt enable (only applicable to STM32F0X2 USB devices)

**\_\_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:**

- \_\_IT\_\_: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt clear
  - RCC\_IT\_LSERDY: LSE ready interrupt clear
  - RCC\_IT\_HSIRDY: HSI ready interrupt clear
  - RCC\_IT\_HSERDY: HSE ready interrupt clear
  - RCC\_IT\_PLLRDY: PLL ready interrupt clear
  - RCC\_IT\_HSI14RDY: HSI14 ready interrupt clear
  - RCC\_IT\_HSI48RDY: HSI48 ready interrupt clear (only applicable to STM32F0X2 USB devices)
  - RCC\_IT\_CSS: Clock Security System interrupt clear

**\_\_HAL\_RCC\_GET\_IT****Description:**

- Check the RCC's interrupt has occurred or not.

**Parameters:**

- \_\_IT\_\_: specifies the RCC interrupt source to check. This parameter can be one of the following values:
  - 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\_HSI14RDY: HSI14 ready interrupt flag
  - RCC\_IT\_HSI48RDY: HSI48 ready interrupt flag (only applicable to STM32F0X2 USB devices)
  - RCC\_IT\_CSS: Clock Security System interrupt flag

**Return value:**

- The new state of IT (TRUE or FALSE).

\_HAL\_RCC\_CLEAR\_RESET\_FLAGS  
RCC\_FLAG\_MASK

**Description:**

- Check RCC flag is set or not.

**Parameters:**

- FLAG: specifies the flag to check. This parameter can be one of the following values:
  - RCC\_FLAG\_HSIRDY: HSI oscillator clock ready
  - RCC\_FLAG\_HSERDY: HSE oscillator clock ready
  - RCC\_FLAG\_PLLRDY: PLL clock ready
  - RCC\_FLAG\_HSI14RDY: HSI14 oscillator clock ready
  - RCC\_FLAG\_HSI48RDY: HSI48 oscillator clock ready (only applicable to STM32F0X2 USB devices)
  - RCC\_FLAG\_LSERDY: LSE oscillator clock ready
  - RCC\_FLAG\_LSIRDY: LSI oscillator clock ready
  - RCC\_FLAG\_OBLRST: Option Byte Load reset
  - RCC\_FLAG\_PINRST: Pin reset
  - RCC\_FLAG\_PORRST: POR/PDR reset
  - RCC\_FLAG\_SFTRST: Software reset
  - RCC\_FLAG\_IWDGRST: Independent Watchdog reset
  - RCC\_FLAG\_WWDGRST: Window Watchdog reset
  - RCC\_FLAG\_LPWRRST: Low Power reset

**Return value:**

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

`__HAL_RCC_GET_FLAG`

**RCC Force Release Backup**

`__HAL_RCC_BACKUPRESET_FORCE`

`__HAL_RCC_BACKUPRESET_RELEASE`

**RCC Get Clock source**

`__HAL_RCC_GET_SYSCLK_SOU` **Description:**  
RCE

- 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 value:
  - 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_OSCSOU` **Description:**  
RCE

- Macro to get the oscillator used as PLL clock source.

**Return value:**

- The: oscillator used as PLL clock source. The returned value can be one of the following:
  - RCC\_PLLSOURCE\_HSI: HSI oscillator is used as PLL clock source.
  - RCC\_PLLSOURCE\_HSE: HSE oscillator is used as PLL clock source.

**RCC HSE Config**

`RCC_HSE_OFF`

`RCC_HSE_ON`

`RCC_HSE_BYPASS`

`IS_RCC_HSE`

**RCC 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

<u>_HAL_RCC_HSE_PREDIV_CONFIG</u>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Macro to configure the External High Speed oscillator (HSE) Predivision factor for PLL.</li> </ul>

**Parameters:**

- \_HSEPredivValue: specifies the division value applied to HSE. This parameter must be a number between RCC\_HSE\_PREDIV\_DIV1 and RCC\_HSE\_PREDIV\_DIV16.

**RCC HSI14 Config**

RCC\_HSI14\_OFF  
RCC\_HSI14\_ON  
RCC\_HSI14\_ADC\_CONTROL  
IS\_RCC\_HSI14  
RCC\_HSI14CALIBRATION\_DEFAULT

**RCC\_HSI14\_Configuration**

\_HAL\_RCC\_HSI14\_ENABLE  
\_HAL\_RCC\_HSI14\_DISABLE  
\_HAL\_RCC\_HSI14ADC\_ENABLE  
\_HAL\_RCC\_HSI14ADC\_DISABLE

RCC\_CR2\_HSI14TRIM\_BitNumber

**Description:**

- Macro to adjust the Internal 14Mhz High Speed oscillator (HSI) calibration value.

**Parameters:**

- \_HSI14CalibrationValue: specifies the calibration trimming value (default is RCC\_HSI14CALIBRATION\_DEFAULT). This parameter must be a number between 0 and 0x1F.

\_HAL\_RCC\_HSI14\_CALIBRATIONVALUE\_ADJ  
UST

**RCC HSI Config**

RCC\_HSI\_OFF  
RCC\_HSI\_ON

`IS_RCC_HSI``RCC_HSICALIBRATION_DEFAULT`***RCC HSI Configuration***`_HAL_RCC_HSI_ENABLE``_HAL_RCC_HSI_DISABLE``RCC_CR_HSITRIM_BitNumber`**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.

`_HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST`***RCC I2C1 Clock Source***`RCC_I2C1CLKSOURCE_HSI``RCC_I2C1CLKSOURCE_SYSCLK``IS_RCC_I2C1CLKSOURCE`***RCC I2Cx Clock Config***`_HAL_RCC_I2C1_CONFIG`**Description:**

- Macro to configure the I2C1 clock (I2C1CLK).

**Parameters:**

- `_I2C1CLKSource_`: specifies the I2C1 clock source. This parameter can be one of the following values:
  - `RCC_I2C1CLKSOURCE_HSI`: HSI selected as I2C1 clock
  - `RCC_I2C1CLKSOURCE_SYSCLK`: System Clock selected as I2C1 clock

`_HAL_RCC_GET_I2C1_SOURCE`**Description:**

- Macro to get the I2C1 clock source.

**Return value:**

- The clock source can be one of the following values:
  - `RCC_I2C1CLKSOURCE_HSI`: HSI selected as I2C1 clock
  - `RCC_I2C1CLKSOURCE_SYSCLK`: System Clock selected as I2C1 clock

***RCC Interrupt***

RCC\_IT\_LSIRDY

RCC\_IT\_LSERDY

RCC\_IT\_HSIRDY

RCC\_IT\_HSERDY

RCC\_IT\_PLLRDY

RCC\_IT\_HSI14

RCC\_IT\_CSS

#### ***RCC\_LSE\_Config***

RCC\_LSE\_OFF

RCC\_LSE\_ON

RCC\_LSE\_BYPASS

IS\_RCC\_LSE

#### ***RCC LSE Configuration***

\_\_HAL\_RCC\_LSE\_CONFIG **Description:**

- Macro to configure the External Low Speed oscillator (LSE).

#### **Parameters:**

- \_\_STATE\_\_: specifies the new state of the LSE. This parameter can be one of the following values:
  - RCC\_LSE\_OFF: turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
  - RCC\_LSE\_ON: turn ON the LSE oscillator
  - RCC\_LSE\_BYPASS: LSE oscillator bypassed with external clock

#### ***RCC LSI Config***

RCC\_LSI\_OFF

RCC\_LSI\_ON

IS\_RCC\_LSI

#### ***RCC LSI Configuration***

\_\_HAL\_RCC\_LSI\_ENABLE

\_\_HAL\_RCC\_LSI\_DISABLE

#### ***RCC MC0x Index***

RCC\_MCO

IS\_RCC\_MCO

#### ***RCC MCO Clock Source***

RCC\_MCOSOURCE\_NONE

RCC\_MCOSOURCE\_LSI

RCC\_MCOSOURCE\_LSE

RCC\_MCOSOURCE\_SYSCLK  
RCC\_MCOSOURCE\_HSI  
RCC\_MCOSOURCE\_HSE  
RCC\_MCOSOURCE\_PLLCLK\_DIV2  
RCC\_MCOSOURCE\_HSI14

***RCC Oscillator Type***

RCC\_OSCILLATORTYPE\_NONE  
RCC\_OSCILLATORTYPE\_HSE  
RCC\_OSCILLATORTYPE\_HSI  
RCC\_OSCILLATORTYPE\_LSE  
RCC\_OSCILLATORTYPE\_LSI  
RCC\_OSCILLATORTYPE\_HSI14  
RCC\_OSCILLATORTYPE\_HSI48  
IS\_RCC\_OSCILLATORTYPE

***RCC PLL Clock Source***

RCC\_PLLSOURCE\_HSE

***RCC PLL Config***

RCC\_PLL\_NONE  
RCC\_PLL\_OFF  
RCC\_PLL\_ON  
IS\_RCC\_PLL

***RCC PLL Configuration***

\_\_HAL\_RCC\_PLL\_ENABLE  
\_\_HAL\_RCC\_PLL\_DISABLE  
\_\_HAL\_RCC\_PLL\_CONFIG

**Description:**

- Macro to configure the PLL clock source, multiplication and division factors.

**Parameters:**

- \_\_RCC\_PLLSource\_\_: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC\_PLLSOURCE\_HSI: HSI oscillator clock selected as PLL clock entry
  - RCC\_PLLSOURCE\_HSE: HSE oscillator clock selected as PLL clock entry
- \_\_PREDIV\_\_: specifies the predivider factor for PLL VCO input clock This parameter must be a number between RCC\_PREDIV\_DIV1 and RCC\_PREDIV\_DIV16.
- \_\_PLLMUL\_\_: specifies the multiplication factor for PLL VCO input clock This parameter must be a

number between RCC\_PLL\_MUL2 and  
RCC\_PLL\_MUL16.

**RCC 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  
IS\_RCC\_PLL\_MUL

**RCC PLL Prediv Factor**

RCC\_PREDIV\_DIV1  
RCC\_PREDIV\_DIV2  
RCC\_PREDIV\_DIV3  
RCC\_PREDIV\_DIV4  
RCC\_PREDIV\_DIV5  
RCC\_PREDIV\_DIV6  
RCC\_PREDIV\_DIV7  
RCC\_PREDIV\_DIV8  
RCC\_PREDIV\_DIV9  
RCC\_PREDIV\_DIV10  
RCC\_PREDIV\_DIV11  
RCC\_PREDIV\_DIV12  
RCC\_PREDIV\_DIV13  
RCC\_PREDIV\_DIV14  
RCC\_PREDIV\_DIV15  
RCC\_PREDIV\_DIV16  
IS\_RCC\_PREDIV

**RCC Private Define**

RCC\_CFGR\_HPRE\_BITNUMBER  
RCC\_CFGR\_PPRE\_BITNUMBER

**RCC Private Macros**

\_MCO\_CLK\_ENABLE  
MCO\_GPIO\_PORT  
MCO\_PIN

**RCC RTC Clock Configuration**

\_HAL\_RCC\_RTC\_ENABLE  
\_HAL\_RCC\_RTC\_DISABLE  
\_HAL\_RCC\_RTC\_CONFIG

**Description:**

- Macro to configure the RTC clock (RTCCLK).

**Parameters:**

- \_\_RTCCLKSource\_\_: specifies the RTC clock source. This parameter can be one of the following values:
  - RCC\_RTCCLKSOURCE\_NONE: No clock selected as RTC clock
  - RCC\_RTCCLKSOURCE\_LSE: LSE selected as RTC clock
  - RCC\_RTCCLKSOURCE\_LSI: LSI selected as RTC clock
  - RCC\_RTCCLKSOURCE\_HSE\_DIV32: HSE clock divided by 32

\_HAL\_RCC\_GET\_RTC\_SOURCE

**Description:**

- Macro to get the RTC clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_RTCCLKSOURCE\_NONE: No clock selected as RTC clock
  - RCC\_RTCCLKSOURCE\_LSE: LSE selected as RTC clock
  - RCC\_RTCCLKSOURCE\_LSI: LSI selected as RTC clock
  - RCC\_RTCCLKSOURCE\_HSE\_DIV32: HSE clock divided by 32 selected as RTC clock

**RCC RTC Clock Source**

RCC\_RTCCLKSOURCE\_NONE  
RCC\_RTCCLKSOURCE\_LSE  
RCC\_RTCCLKSOURCE\_LSI  
RCC\_RTCCLKSOURCE\_HSE\_DIV32

IS\_RCC\_RTCCLKSOURCE  
**RCC System Clock Source**  
RCC\_SYSCLKSOURCE\_HSI  
RCC\_SYSCLKSOURCE\_HSE  
RCC\_SYSCLKSOURCE\_PLLCLK  
**RCC System Clock Source Status**  
RCC\_SYSCLKSOURCE\_STATUS\_HSI  
RCC\_SYSCLKSOURCE\_STATUS\_HSE  
RCC\_SYSCLKSOURCE\_STATUS\_PLLCLK  
**RCC System Clock Type**  
RCC\_CLOCKTYPE\_SYSCLK  
RCC\_CLOCKTYPE\_HCLK  
RCC\_CLOCKTYPE\_PCLK1  
IS\_RCC\_CLOCKTYPE  
**RCC Timeout**  
LSE\_TIMEOUT\_VALUE  
LSE\_TIMEOUT\_VALUE  
DBP\_TIMEOUT\_VALUE  
HSE\_TIMEOUT\_VALUE  
HSI\_TIMEOUT\_VALUE  
LSI\_TIMEOUT\_VALUE  
HSI14\_TIMEOUT\_VALUE  
HSI48\_TIMEOUT\_VALUE  
PLL\_TIMEOUT\_VALUE  
CLOCKSWITCH\_TIMEOUT\_VALUE  
**RCC USART1 Clock Source**  
RCC\_USART1CLKSOURCE\_PCLK1  
RCC\_USART1CLKSOURCE\_SYSCLK  
RCC\_USART1CLKSOURCE\_LSE  
RCC\_USART1CLKSOURCE\_HSI  
IS\_RCC\_USART1CLKSOURCE  
**RCC USARTx Clock Config**

`_HAL_RCC_USART1_CONFIG`**Description:**

- Macro to configure the USART1 clock (USART1CLK).

**Parameters:**

- `_USART1CLKSource_`: specifies the

USART1 clock source. This parameter can be one of the following values:

- RCC\_USART1CLKSOURCE\_PCLK1: PCLK1 selected as USART1 clock
- RCC\_USART1CLKSOURCE\_HSI: HSI selected as USART1 clock
- RCC\_USART1CLKSOURCE\_SYSCLK: System Clock selected as USART1 clock
- RCC\_USART1CLKSOURCE\_LSE: LSE selected as USART1 clock

`_HAL_RCC_GET_USART1_SOURCE`

`CE`

**Description:**

- Macro to get the USART1 clock source.

**Return value:**

- The clock source can be one of the following values:
  - RCC\_USART1CLKSOURCE\_PCLK1: PCLK1 selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_HSI: HSI selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_SYSCLK: System Clock selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_LSE: LSE selected as USART1 clock

## 31 HAL RCC Extension Driver

### 31.1 RCCEEx Firmware driver registers structures

#### 31.1.1 RCC\_PeriphCLKInitTypeDef

*RCC\_PeriphCLKInitTypeDef* is defined in the `stm32f0xx_hal_rcc_ex.h`

##### Data Fields

- *uint32\_t PeriphClockSelection*
- *uint32\_t RTCClockSelection*
- *uint32\_t Usart1ClockSelection*
- *uint32\_t Usart2ClockSelection*
- *uint32\_t Usart3ClockSelection*
- *uint32\_t I2c1ClockSelection*
- *uint32\_t CecClockSelection*

##### Field Documentation

- *uint32\_t RCC\_PeriphCLKInitTypeDef::PeriphClockSelection* The Extended Clock to be configured. This parameter can be a value of [\*RCCEEx\\_Periph\\_Clock\\_Selection\*](#)
- *uint32\_t RCC\_PeriphCLKInitTypeDef::RTCClockSelection* Specifies RTC Clock Prescalers Selection This parameter can be a value of [\*RCC\\_RTC\\_Clock\\_Source\*](#)
- *uint32\_t RCC\_PeriphCLKInitTypeDef::Usart1ClockSelection* USART1 clock source This parameter can be a value of [\*RCC\\_USART1\\_Clock\\_Source\*](#)
- *uint32\_t RCC\_PeriphCLKInitTypeDef::Usart2ClockSelection* USART2 clock source This parameter can be a value of [\*RCCEEx\\_USART2\\_Clock\\_Source\*](#)
- *uint32\_t RCC\_PeriphCLKInitTypeDef::Usart3ClockSelection* USART3 clock source This parameter can be a value of [\*RCCEEx\\_USART3\\_Clock\\_Source\*](#)
- *uint32\_t RCC\_PeriphCLKInitTypeDef::I2c1ClockSelection* I2C1 clock source This parameter can be a value of [\*RCC\\_I2C1\\_Clock\\_Source\*](#)
- *uint32\_t RCC\_PeriphCLKInitTypeDef::CecClockSelection* HDMI CEC clock source This parameter can be a value of [\*RCCEEx\\_CEC\\_Clock\\_Source\*](#)

#### 31.1.2 RCC\_CRSInitTypeDef

*RCC\_CRSInitTypeDef* is defined in the `stm32f0xx_hal_rcc_ex.h`

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Source*
- *uint32\_t Polarity*
- *uint32\_t ReloadValue*
- *uint32\_t ErrorLimitValue*
- *uint32\_t HSI48CalibrationValue*

##### Field Documentation

- ***uint32\_t RCC\_CRSInitTypeDef::Prescaler*** Specifies the division factor of the SYNC signal. This parameter can be a value of ***RCCEEx\_CRS\_SynchroDivider***
- ***uint32\_t RCC\_CRSInitTypeDef::Source*** Specifies the SYNC signal source. This parameter can be a value of ***RCCEEx\_CRS\_SynchroSource***
- ***uint32\_t RCC\_CRSInitTypeDef::Polarity*** Specifies the input polarity for the SYNC signal source. This parameter can be a value of ***RCCEEx\_CRS\_SynchroPolarity***
- ***uint32\_t RCC\_CRSInitTypeDef::ReloadValue*** Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro ***\_HAL\_RCC\_CRS\_CALCULATE\_RELOADVALUE(\_FTARGET\_, \_FSYNC\_)*** This parameter must be a number between 0 and 0xFFFF or a value of ***RCCEEx\_CRS\_ReloadValueDefault***.
- ***uint32\_t RCC\_CRSInitTypeDef::ErrorLimitValue*** Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of ***RCCEEx\_CRS\_ErrorLimitDefault***
- ***uint32\_t RCC\_CRSInitTypeDef::HSI48CalibrationValue*** Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x3F or a value of ***RCCEEx\_CRS\_HSI48CalibrationDefault***

### 31.1.3 RCC\_CRSSynchroInfoTypeDef

**RCC\_CRSSynchroInfoTypeDef** is defined in the `stm32f0xx_hal_rcc_ex.h`

#### Data Fields

- ***uint32\_t ReloadValue***
- ***uint32\_t HSI48CalibrationValue***
- ***uint32\_t FreqErrorCapture***
- ***uint32\_t FreqErrorDirection***

#### Field Documentation

- ***uint32\_t RCC\_CRSSynchroInfoTypeDef::ReloadValue*** Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFF
- ***uint32\_t RCC\_CRSSynchroInfoTypeDef::HSI48CalibrationValue*** Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x3F
- ***uint32\_t RCC\_CRSSynchroInfoTypeDef::FreqErrorCapture*** Specifies the value loaded in the .FECAP, the frequency error counter value latched in the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFF
- ***uint32\_t RCC\_CRSSynchroInfoTypeDef::FreqErrorDirection*** Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of ***RCCEEx\_CRS\_FreqErrorDirection***

## 31.2 RCCEEx Firmware driver API description

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

### 31.2.1 How to use this driver

### 31.2.2 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\_RCCEEx\_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\_RCC\_OscConfig\(\)`](#)
- [`HAL\_RCC\_ClockConfig\(\)`](#)
- [`HAL\_RCC\_GetSysClockFreq\(\)`](#)
- [`HAL\_RCCEEx\_PeriphCLKConfig\(\)`](#)
- [`HAL\_RCCEEx\_GetPeriphCLKConfig\(\)`](#)
- [`HAL\_RCCEEx\_CRSConfig\(\)`](#)
- [`HAL\_RCCEEx\_CRSSoftwareSynchronizationGenerate\(\)`](#)
- [`HAL\_RCCEEx\_CRSGetSynchronizationInfo\(\)`](#)
- [`HAL\_RCCEEx\_CRSWaitSynchronization\(\)`](#)

### 31.2.3 HAL\_RCC\_OscConfig

Function Name	<code>HAL_StatusTypeDef HAL_RCC_OscConfig (</code> <code>RCC_OscInitTypeDef * RCC_OscInitStruct)</code>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The PLL is not disabled when used as system clock.</li> </ul>

### 31.2.4 HAL\_RCC\_ClockConfig

Function Name	<code>HAL_StatusTypeDef HAL_RCC_ClockConfig (</code> <code>RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)</code>
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_ClkInitTypeDef 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:           <ul style="list-style-type: none"> <li>– <b>FLASH_LATENCY_0</b> FLASH 0 Latency cycle</li> <li>– <b>FLASH_LATENCY_1</b> FLASH 1 Latency cycle</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></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.</li> </ul>

### 31.2.5 HAL\_RCC\_GetSysClockFreq

Function Name	<code>uint32_t HAL_RCC_GetSysClockFreq ( void )</code>
Function Description	Returns the SYSCLK frequency.
Return values	<ul style="list-style-type: none"> <li><b>SYSCLK frequency</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>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:</li> <li>If SYSCLK source is HSI, function returns a value based on HSI_VALUE(*)</li> <li>If SYSCLK source is HSI48, function returns a value based on HSI48_VALUE(*)</li> <li>If SYSCLK source is HSE, function returns a value based on HSE_VALUE divided by PREDIV factor(**)</li> <li>If SYSCLK source is PLL, function returns a value based on HSE_VALUE divided by PREDIV factor(**) or depending on STM32F0xx devices either a value based on HSI_VALUE divided by 2 or HSI_VALUE divided by PREDIV factor(*) multiplied by the PLL factor .</li> <li>(*) HSI_VALUE &amp; HSI48_VALUE are constants defined in stm32f0xx_hal_conf.h file (default values 8 MHz and 48MHz).</li> <li>(**) HSE_VALUE is a constant defined in stm32f0xx_hal_conf.h file (default value 8 MHz), user has to</li> </ul>

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.

### 31.2.6 HAL\_RCCEEx\_PeriphCLKConfig

Function Name	<code>HAL_StatusTypeDef HAL_RCCEEx_PeriphCLKConfig ( RCC_PeriphCLKInitTypeDef * PeriphClkInit)</code>
Function Description	Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClkInit</b> : pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks (USART, RTC, I2C, CEC and USB).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Care must be taken when HAL_RCCEEx_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.</li> </ul>

### 31.2.7 HAL\_RCCEEx\_GetPeriphCLKConfig

Function Name	<code>void HAL_RCCEEx_GetPeriphCLKConfig ( RCC_PeriphCLKInitTypeDef * PeriphClkInit)</code>
Function Description	Get the RCC_ClkInitStruct 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 (USART, RTC, I2C, CEC and</li> </ul>

---

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

### 31.2.8 HAL\_RCCEEx\_CRSConfig

Function Name	<b>void HAL_RCCEEx_CRSConfig ( <i>RCC_CRSInitTypeDef</i> * pInit)</b>
Function Description	Start automatic synchronization using polling mode.
Parameters	<ul style="list-style-type: none"><li><b>pInit</b> : Pointer on RCC_CRSInitTypeDef structure</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 31.2.9 HAL\_RCCEEx\_CRSSoftwareSynchronizationGenerate

Function Name	<b>void HAL_RCCEEx_CRSSoftwareSynchronizationGenerate ( void )</b>
Function Description	Generate the software synchronization event.
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 31.2.10 HAL\_RCCEEx\_CRSGetSynchronizationInfo

Function Name	<b>void HAL_RCCEEx_CRSGetSynchronizationInfo ( <i>RCC_CRSSynchroInfoTypeDef</i> * pSynchroInfo)</b>
Function Description	Function to return synchronization info.
Parameters	<ul style="list-style-type: none"><li><b>pSynchroInfo</b> : Pointer on RCC_CRSSynchroInfoTypeDef structure</li></ul>

---

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

### 31.2.11 HAL\_RCCEEx\_CRSWaitSynchronization

Function Name	<b>RCC_CRSStatusTypeDef</b> <b>HAL_RCCEEx_CRSWaitSynchronization ( uint32_t Timeout)</b>
Function Description	This function handles CRS Synchronization Timeout.
Parameters	<ul style="list-style-type: none"> <li>• <b>Timeout</b> : Duration of the timeout</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Combination of Synchronization status This parameter can be a combination of the following values:</b> <ul style="list-style-type: none"> <li>- <b>RCC_CRS_TIMEOUT</b></li> <li>- <b>RCC_CRS_SYNCOK</b></li> <li>- <b>RCC_CRS_SYNCWARM</b></li> <li>- <b>RCC_CRS_SYNCERR</b></li> <li>- <b>RCC_CRS_SYNCMISS</b></li> <li>- <b>RCC_CRS_TRIMOV</b></li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency.</li> <li>• If Timeout set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.</li> </ul>

## 31.3 RCCEEx Firmware driver defines

### 31.3.1 RCCEEx

RCCEEx

**RCCEEx CEC Clock Source**

RCC\_CECCLKSOURCE\_HSI

RCC\_CECCLKSOURCE\_LSE

IS\_RCC\_CECCLKSOURCE

**RCCEEx CRS ErrorLimitDefault**

RCC\_CRS\_ERRORLIMIT\_DEFAULT Default Frequency error limit

IS\_RCC\_CRS\_ERRORLIMIT

**RCCEEx CRS Extended Features**

---

`_HAL_RCC_CRS_ENABLE_FREQ_ERROR_COUNTER`**Description:**

- Enables the oscillator clock for frequency error counter.

**Return value:**

- None:

`_HAL_RCC_CRS_DISABLE_FREQ_ERROR_COUNTER`**Description:**

- Disables the oscillator clock for frequency error counter.

**Return value:**

- None:

`_HAL_RCC_CRS_ENABLE_AUTOMATIC_CALIB`**Description:**

- Enables the automatic hardware adjustement of TRIM bits.

**Return value:**

- None:

`_HAL_RCC_CRS_DISABLE_AUTOMATIC_CALIB`**Description:**

- Enables or disables the automatic hardware adjustement of TRIM bits.

**Return value:**

- None:

`_HAL_RCC_CRS_CALCULATE_RELOADVALUE`**Description:**

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

**Parameters:**

- `_FTARGET_`: Target frequency (value in Hz)
- `_FSYNC_`: Synchronization signal frequency (value in Hz)

**Return value:**

- None:

***RCCEx CRS Flags***`RCC_CRS_FLAG_SYNCOK``RCC_CRS_FLAG_SYNCWARN`

RCC_CRS_FLAG_ERR	
RCC_CRS_FLAG_ESYNC	
RCC_CRS_FLAG_TRIMOVF	Trimming overflow or underflow
RCC_CRS_FLAG_SYNCERR	SYNC error
RCC_CRS_FLAG_SYNCMISS	SYNC missed
<b><i>RCCEx CRS FreqErrorDirection</i></b>	
RCC_CRS_FREQERRORDIR_UP	Upcounting direction, the actual frequency is above the target
RCC_CRS_FREQERRORDIR_DOWN	Downcounting direction, the actual frequency is below the target
IS_RCC_CRS_FREQERRORDIR	
<b><i>RCCEx CRS HSI48CalibrationDefault</i></b>	
RCC_CRS_HSI48CALIBRATION_DEFAULT	The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency
IS_RCC_CRS_HSI48CALIBRATION	
<b><i>RCCEx CRS Interrupt Sources</i></b>	
RCC_CRS_IT_SYNCOK	SYNC event OK
RCC_CRS_IT_SYNCWARN	SYNC warning
RCC_CRS_IT_ERR	error
RCC_CRS_IT_ESYNC	Expected SYNC
RCC_CRS_IT_TRIMOVF	Trimming overflow or underflow
RCC_CRS_IT_SYNCERR	SYNC error
RCC_CRS_IT_SYNCMISS	SYNC missed
<b><i>RCCEx CRS ReloadValueDefault</i></b>	
RCC_CRS_RELOADVALUE_DEFAULT	The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).
IS_RCC_CRS_RELOADVALUE	
<b><i>RCCEx CRS SynchroDivider</i></b>	
RCC_CRS_SYNC_DIV1	Synchro Signal not divided (default)
RCC_CRS_SYNC_DIV2	Synchro Signal divided by 2
RCC_CRS_SYNC_DIV4	Synchro Signal divided by 4
RCC_CRS_SYNC_DIV8	Synchro Signal divided by 8
RCC_CRS_SYNC_DIV16	Synchro Signal divided by 16
RCC_CRS_SYNC_DIV32	Synchro Signal divided by 32

RCC\_CRS\_SYNC\_DIV64    Synchro Signal divided by 64

RCC\_CRS\_SYNC\_DIV128    Synchro Signal divided by 128

IS\_RCC\_CRS\_SYNC\_DIV

***RCCEx CRS SyncroPolarity***

RCC\_CRS\_SYNC\_POLARITY\_RISING    Synchro Active on rising edge (default)

RCC\_CRS\_SYNC\_POLARITY\_FALLING    Synchro Active on falling edge

IS\_RCC\_CRS\_SYNC\_POLARITY

***RCCEx CRS SyncroSource***

RCC\_CRS\_SYNC\_SOURCE\_GPIO    Synchro Signal source GPIO

RCC\_CRS\_SYNC\_SOURCE\_LSE    Synchro Signal source LSE

RCC\_CRS\_SYNC\_SOURCE\_USB    Synchro Signal source USB SOF (default)

IS\_RCC\_CRS\_SYNC\_SOURCE

***RCCEx Force Release Peripheral Reset***

\_\_GPIOD\_FORCE\_RESET

\_\_GPIOD\_RELEASE\_RESET

\_\_GPIOE\_FORCE\_RESET

\_\_GPIOE\_RELEASE\_RESET

\_\_TSC\_FORCE\_RESET

\_\_TSC\_RELEASE\_RESET

\_\_USART2\_FORCE\_RESET

\_\_SPI2\_FORCE\_RESET

\_\_USART2\_RELEASE\_RESET

\_\_SPI2\_RELEASE\_RESET

\_\_TIM2\_FORCE\_RESET

\_\_TIM2\_RELEASE\_RESET

\_\_TIM6\_FORCE\_RESET

\_\_I2C2\_FORCE\_RESET

\_\_TIM6\_RELEASE\_RESET

\_\_I2C2\_RELEASE\_RESET

\_\_DAC1\_FORCE\_RESET

\_\_DAC1\_RELEASE\_RESET

\_\_CEC\_FORCE\_RESET

\_\_CEC\_RELEASE\_RESET

\_\_TIM7\_FORCE\_RESET

\_\_USART3\_FORCE\_RESET

\_\_USART4\_FORCE\_RESET

---

```

__TIM7_RELEASE_RESET
__USART3_RELEASE_RESET
__USART4_RELEASE_RESET
__CAN_FORCE_RESET
__CAN_RELEASE_RESET
__CRS_FORCE_RESET
__CRS_RELEASE_RESET
__USART5_FORCE_RESET
__USART5_RELEASE_RESET
__TIM15_FORCE_RESET
__TIM15_RELEASE_RESET
__USART6_FORCE_RESET
__USART6_RELEASE_RESET
__USART7_FORCE_RESET
__USART8_FORCE_RESET
__USART7_RELEASE_RESET
__USART8_RELEASE_RESET

```

**RCCEx HSI48 Config**

RCC\_HSI48\_OFF

RCC\_HSI48\_ON

IS\_RCC\_HSI48

**RCCEx HSI48 Enable Disable**

```

__HAL_RCC_HSI48_ENABLE
__HAL_RCC_HSI48_DISABLE
__HAL_RCC_GET_HSI48_STATE

```

**Description:**

- Macro to get the Internal 48Mhz High Speed oscillator (HSI48) state.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_HSI48\_ON: HSI48 enabled
  - RCC\_HSI48\_OFF: HSI48 disabled

**RCCEx IT and Flag**

`__HAL_RCC_CRS_ENABLE_IT`

**Description:**

- Enables the specified CRS interrupts.

**Parameters:**

- `__INTERRUPT__`: specifies the CRS interrupt sources to be enabled. This parameter can be any combination of the

following values:

- RCC\_CRS\_IT\_SYNCOK
- RCC\_CRS\_IT\_SYNCWARN
- RCC\_CRS\_IT\_ERR
- RCC\_CRS\_IT\_ESYNC

**Return value:**

- None:

`_HAL_RCC_CRS_DISABLE_IT`

**Description:**

- Disables the specified CRS interrupts.

**Parameters:**

- `_INTERRUPT_`: specifies the CRS interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC\_CRS\_IT\_SYNCOK
  - RCC\_CRS\_IT\_SYNCWARN
  - RCC\_CRS\_IT\_ERR
  - RCC\_CRS\_IT\_ESYNC

**Return value:**

- None:

`_HAL_RCC_CRS_GET_IT_SOURCE`

**Description:**

- Check the CRS's interrupt has occurred or not.

**Parameters:**

- `_INTERRUPT_`: specifies the CRS interrupt source to check. This parameter can be one of the following values:
  - RCC\_CRS\_IT\_SYNCOK
  - RCC\_CRS\_IT\_SYNCWARN
  - RCC\_CRS\_IT\_ERR
  - RCC\_CRS\_IT\_ESYNC

**Return value:**

- The: new state of `_INTERRUPT_` (SET or RESET).

`RCC_CRS_IT_ERROR_MASK`

**Description:**

- Clear the CRS's interrupt pending bits 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\_CRS\_IT\_SYNCOK
  - RCC\_CRS\_IT\_SYNCWARN
  - RCC\_CRS\_IT\_ERR
  - RCC\_CRS\_IT\_ESYNC

- RCC\_CRS\_IT\_TRIMOVF
- RCC\_CRS\_IT\_SYNCERR
- RCC\_CRS\_IT\_SYNCMISS

`_HAL_RCC_CRS_CLEAR_IT`  
`_HAL_RCC_CRS_GET_FLAG`

**Description:**

- Checks whether the specified CRS flag is set or not.

**Parameters:**

- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
  - RCC\_CRS\_FLAG\_SYNCOK
  - RCC\_CRS\_FLAG\_SYNCWARN
  - RCC\_CRS\_FLAG\_ERR
  - RCC\_CRS\_FLAG\_ESYNC
  - RCC\_CRS\_FLAG\_TRIMOVF
  - RCC\_CRS\_FLAG\_SYNCERR
  - RCC\_CRS\_FLAG\_SYNCMISS

**Return value:**

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

`RCC_CRS_FLAG_ERROR_MASK`

**Description:**

- Clears the CRS specified FLAG.

**Parameters:**

- `_FLAG_`: specifies the flag to clear. This parameter can be one of the following values:
  - RCC\_CRS\_FLAG\_SYNCOK
  - RCC\_CRS\_FLAG\_SYNCWARN
  - RCC\_CRS\_FLAG\_ERR
  - RCC\_CRS\_FLAG\_ESYNC
  - RCC\_CRS\_FLAG\_TRIMOVF
  - RCC\_CRS\_FLAG\_SYNCERR
  - RCC\_CRS\_FLAG\_SYNCMISS

**Return value:**

- None:

`_HAL_RCC_CRS_CLEAR_FLAG`

***RCCEEx MCOx Clock Prescaler***

`RCC_MCO_DIV1`  
`RCC_MCO_DIV2`  
`RCC_MCO_DIV4`  
`RCC_MCO_DIV8`  
`RCC_MCO_DIV16`

```
RCC_MCO_DIV32
RCC_MCO_DIV64
RCC_MCO_DIV128
IS_RCC_MCODIV
RCCEx MCO Clock Source
RCC_MCOSOURCE_HSI48
RCC_MCOSOURCE_PLLCLK_NODIV
IS_RCC_MCOSOURCE
RCC_IT_HSI48
RCC_CR2_HSI48RDY_BitNumber
RCC_FLAG_HSI48RDY
RCCEx_Peripheral_Clock_Enable_Disable
__GPIOD_CLK_ENABLE
__GPIOD_CLK_DISABLE
__GPIOE_CLK_ENABLE
__GPIOE_CLK_DISABLE
__TSC_CLK_ENABLE
__TSC_CLK_DISABLE
__DMA2_CLK_ENABLE
__DMA2_CLK_DISABLE
__USART2_CLK_ENABLE
__SPI2_CLK_ENABLE
__USART2_CLK_DISABLE
__SPI2_CLK_DISABLE
__TIM2_CLK_ENABLE
__TIM2_CLK_DISABLE
__TIM6_CLK_ENABLE
__I2C2_CLK_ENABLE
__TIM6_CLK_DISABLE
__I2C2_CLK_DISABLE
__DAC1_CLK_ENABLE
__DAC1_CLK_DISABLE
__CEC_CLK_ENABLE
__CEC_CLK_DISABLE
__TIM7_CLK_ENABLE
__USART3_CLK_ENABLE
```

---

\_\_USART4\_CLK\_ENABLE  
 \_\_TIM7\_CLK\_DISABLE  
 \_\_USART3\_CLK\_DISABLE  
 \_\_USART4\_CLK\_DISABLE  
 \_\_CAN\_CLK\_ENABLE  
 \_\_CAN\_CLK\_DISABLE  
 \_\_CRS\_CLK\_ENABLE  
 \_\_CRS\_CLK\_DISABLE  
 \_\_USART5\_CLK\_ENABLE  
 \_\_USART5\_CLK\_DISABLE  
 \_\_TIM15\_CLK\_ENABLE  
 \_\_TIM15\_CLK\_DISABLE  
 \_\_USART6\_CLK\_ENABLE  
 \_\_USART6\_CLK\_DISABLE  
 \_\_USART7\_CLK\_ENABLE  
 \_\_USART8\_CLK\_ENABLE  
 \_\_USART7\_CLK\_DISABLE  
 \_\_USART8\_CLK\_DISABLE

#### **RCCEEx Peripheral Clock Source Config**

`__HAL_RCC_CEC_CONFIG`

##### **Description:**

- Macro to configure the CEC clock.

##### **Parameters:**

- `__CECCLKSource`: specifies the CEC clock source. This parameter can be one of the following values:
  - `RCC_CECCLKSOURCE_HSI`: HSI selected as CEC clock
  - `RCC_CECCLKSOURCE_LSE`: LSE selected as CEC clock

`__HAL_RCC_GET_CEC_SOURCE`

##### **Description:**

- Macro to get the HDMI CEC clock source.

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

- The: clock source can be one of the following values:
  - `RCC_CECCLKSOURCE_HSI`: HSI selected as CEC clock
  - `RCC_CECCLKSOURCE_LSE`: LSE selected as CEC clock

`__HAL_RCC_MCO_CONFIG`

##### **Description:**

- Macro to configure the MCO clock.

**Parameters:**

- `_MCOClockSource`: specifies the MCO clock source. This parameter can be one of the following values:
  - `RCC_MCOSOURCE_HSI`: HSI selected as MCO clock
  - `RCC_MCOSOURCE_HSE`: HSE selected as MCO clock
  - `RCC_MCOSOURCE_LSI`: LSI selected as MCO clock
  - `RCC_MCOSOURCE_LSE`: LSE selected as MCO clock
  - `RCC_MCOSOURCE_PLLCLK_NODIV`: PLLCLK selected as MCO clock
  - `RCC_MCOSOURCE_PLLCLK_DIV2`: PLLCLK Divided by 2 selected as MCO clock
  - `RCC_MCOSOURCE_SYSCLK`: System Clock selected as MCO clock
  - `RCC_MCOSOURCE_HSI14`: HSI14 selected as MCO clock
  - `RCC_MCOSOURCE_HSI48`: HSI48 selected as MCO clock
- `_MCODiv`: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - `RCC_MCO_DIV1`: MCO clock source is divided by 1
  - `RCC_MCO_DIV2`: MCO clock source is divided by 2
  - `RCC_MCO_DIV4`: MCO clock source is divided by 4
  - `RCC_MCO_DIV8`: MCO clock source is divided by 8
  - `RCC_MCO_DIV16`: MCO clock source is divided by 16
  - `RCC_MCO_DIV32`: MCO clock source is divided by 32
  - `RCC_MCO_DIV64`: MCO clock source is divided by 64
  - `RCC_MCO_DIV128`: MCO clock source is divided by 128

`_HAL_RCC_USART2_CONFIG`**Description:**

- Macro to configure the USART2 clock (USART2CLK).

**Parameters:**

- `_USART2CLKSource`: specifies the USART2 clock source. This parameter can be one of the following values:
  - `RCC_USART2CLKSOURCE_PCLK1`: PCLK1 selected as USART2 clock
  - `RCC_USART2CLKSOURCE_HSI`: HSI

- selected as USART2 clock
- RCC\_USART2CLKSOURCE\_SYSCLK: System Clock selected as USART2 clock
- RCC\_USART2CLKSOURCE\_LSE: LSE selected as USART2 clock

`_HAL_RCC_GET_USART2_SOURCE`

**Description:**

- Macro to get the USART2 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_USART2CLKSOURCE\_PCLK1: PCLK1 selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_HSI: HSI selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_SYSCLK: System Clock selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_LSE: LSE selected as USART2 clock

`_HAL_RCC_USART3_CONFIG`

**Description:**

- Macro to configure the USART3 clock (USART3CLK).

**Parameters:**

- `_USART3CLKSource`: specifies the USART3 clock source. This parameter can be one of the following values:
  - RCC\_USART3CLKSOURCE\_PCLK1: PCLK1 selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_HSI: HSI selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_SYSCLK: System Clock selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_LSE: LSE selected as USART3 clock

`_HAL_RCC_GET_USART3_SOURCE`

**Description:**

- Macro to get the USART3 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_USART3CLKSOURCE\_PCLK1: PCLK1 selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_HSI: HSI selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_SYSCLK: System Clock selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_LSE: LSE selected as USART3 clock

***RCCEx Periph Clock Selection***

RCC\_PERIPHCLK\_USART1  
RCC\_PERIPHCLK\_USART2  
RCC\_PERIPHCLK\_I2C1  
RCC\_PERIPHCLK\_CEC  
RCC\_PERIPHCLK\_RTC  
RCC\_PERIPHCLK\_USART3  
IS\_RCC\_PERIPHCLK  
**RCCEx PLL Clock Source**  
RCC\_PLLSOURCE\_HSI  
RCC\_PLLSOURCE\_HSI48  
IS\_RCC\_PLLSOURCE  
**RCCEx Private Define**  
HSI48\_TIMEOUT\_VALUE  
CRS\_CFGR\_FELIM\_BITNUMBER  
CRS\_CR\_TRIM\_BITNUMBER  
CRS\_ISR FECAP\_BITNUMBER  
**RCCEx System Clock Source**  
RCC\_SYSCLKSOURCE\_HSI48  
IS\_RCC\_SYSCLKSOURCE  
RCC\_SYSCLKSOURCE\_STATUS\_HSI48  
IS\_RCC\_SYSCLKSOURCE\_STATUS  
**RCCEx USART2 Clock Source**  
RCC\_USART2CLKSOURCE\_PCLK1  
RCC\_USART2CLKSOURCE\_SYSCLK  
RCC\_USART2CLKSOURCE\_LSE  
RCC\_USART2CLKSOURCE\_HSI  
IS\_RCC\_USART2CLKSOURCE  
**RCCEx USART3 Clock Source**  
RCC\_USART3CLKSOURCE\_PCLK1  
RCC\_USART3CLKSOURCE\_SYSCLK  
RCC\_USART3CLKSOURCE\_LSE  
RCC\_USART3CLKSOURCE\_HSI  
IS\_RCC\_USART3CLKSOURCE

## 32 HAL RTC Generic Driver

### 32.1 RTC Firmware driver registers structures

#### 32.1.1 RTC\_InitTypeDef

*RTC\_InitTypeDef* is defined in the `stm32f0xx_hal_rtc.h`

##### Data Fields

- *uint32\_t HourFormat*
- *uint32\_t AsynchPrediv*
- *uint32\_t SynchPrediv*
- *uint32\_t OutPut*
- *uint32\_t OutPutPolarity*
- *uint32\_t OutPutType*

##### Field Documentation

- *uint32\_t RTC\_InitTypeDef::HourFormat* Specifies the RTC Hour Format. This parameter can be a value of [\*RTC\\_Hour\\_Formats\*](#)
- *uint32\_t RTC\_InitTypeDef::AsynchPrediv* Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x7F
- *uint32\_t RTC\_InitTypeDef::SynchPrediv* Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x7FFF
- *uint32\_t RTC\_InitTypeDef::OutPut* Specifies which signal will be routed to the RTC output. This parameter can be a value of [\*RTCEx\\_Output\\_selection\\_Definitions\*](#)
- *uint32\_t RTC\_InitTypeDef::OutPutPolarity* Specifies the polarity of the output signal. This parameter can be a value of [\*RTC\\_Output\\_Polarity\\_Definitions\*](#)
- *uint32\_t RTC\_InitTypeDef::OutPutType* Specifies the RTC Output Pin mode. This parameter can be a value of [\*RTC\\_Output\\_Type\\_ALARM\\_OUT\*](#)

#### 32.1.2 RTC\_TimeTypeDef

*RTC\_TimeTypeDef* is defined in the `stm32f0xx_hal_rtc.h`

##### Data Fields

- *uint8\_t Hours*
- *uint8\_t Minutes*
- *uint8\_t Seconds*
- *uint32\_t SubSeconds*
- *uint8\_t TimeFormat*
- *uint32\_t DayLightSaving*
- *uint32\_t StoreOperation*

##### Field Documentation

- ***uint8\_t RTC\_TimeTypeDef::Hours*** Specifies the RTC Time Hour. This parameter must be a number between Min\_Data = 0 and Max\_Data = 12 if the RTC\_HourFormat\_12 is selected. This parameter must be a number between Min\_Data = 0 and Max\_Data = 23 if the RTC\_HourFormat\_24 is selected
- ***uint8\_t RTC\_TimeTypeDef::Minutes*** Specifies the RTC Time Minutes. This parameter must be a number between Min\_Data = 0 and Max\_Data = 59
- ***uint8\_t RTC\_TimeTypeDef::Seconds*** Specifies the RTC Time Seconds. This parameter must be a number between Min\_Data = 0 and Max\_Data = 59
- ***uint32\_t RTC\_TimeTypeDef::SubSeconds*** Specifies the RTC Time SubSeconds. This parameter must be a number between Min\_Data = 0 and Max\_Data = 59
- ***uint8\_t RTC\_TimeTypeDef::TimeFormat*** Specifies the RTC AM/PM Time. This parameter can be a value of [\*RTC\\_AM\\_PM\\_Definitions\*](#)
- ***uint32\_t RTC\_TimeTypeDef::DayLightSaving*** Specifies RTC\_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of [\*RTC\\_DayLightSaving\\_Definitions\*](#)
- ***uint32\_t RTC\_TimeTypeDef::StoreOperation*** Specifies RTC\_StoreOperation value to be written in the BCK bit in CR register to store the operation. This parameter can be a value of [\*RTC\\_StoreOperation\\_Definitions\*](#)

### 32.1.3 RTC\_DateTypeDef

*RTC\_DateTypeDef* is defined in the stm32f0xx\_hal\_rtc.h

#### Data Fields

- ***uint8\_t WeekDay***
- ***uint8\_t Month***
- ***uint8\_t Date***
- ***uint8\_t Year***

#### Field Documentation

- ***uint8\_t RTC\_DateTypeDef::WeekDay*** Specifies the RTC Date WeekDay. This parameter can be a value of [\*RTC\\_WeekDay\\_Definitions\*](#)
- ***uint8\_t RTC\_DateTypeDef::Month*** Specifies the RTC Date Month (in BCD format). This parameter can be a value of [\*RTC\\_Month\\_Date\\_Definitions\*](#)
- ***uint8\_t RTC\_DateTypeDef::Date*** Specifies the RTC Date. This parameter must be a number between Min\_Data = 1 and Max\_Data = 31
- ***uint8\_t RTC\_DateTypeDef::Year*** Specifies the RTC Date Year. This parameter must be a number between Min\_Data = 0 and Max\_Data = 99

### 32.1.4 RTC\_AlarmTypeDef

*RTC\_AlarmTypeDef* is defined in the stm32f0xx\_hal\_rtc.h

#### Data Fields

- ***RTC\_TimeTypeDef AlarmTime***
- ***uint32\_t AlarmMask***
- ***uint32\_t AlarmSubSecondMask***
- ***uint32\_t AlarmDateWeekDaySel***
- ***uint8\_t AlarmDateWeekDay***
- ***uint32\_t Alarm***

### Field Documentation

- ***RTC\_TimeTypeDef RTC\_AlarmTypeDef::AlarmTime*** Specifies the RTC Alarm Time members
- ***uint32\_t RTC\_AlarmTypeDef::AlarmMask*** Specifies the RTC Alarm Masks. This parameter can be a value of [\*\*RTC\\_AlarmMask\\_Definitions\*\*](#)
- ***uint32\_t RTC\_AlarmTypeDef::AlarmSubSecondMask*** Specifies the RTC Alarm SubSeconds Masks. This parameter can be a value of [\*\*RTC\\_Alarm\\_Sub\\_Seconds\\_Masks\\_Definitions\*\*](#)
- ***uint32\_t RTC\_AlarmTypeDef::AlarmDateWeekDaySel*** Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of [\*\*RTC\\_AlarmDateWeekDay\\_Definitions\*\*](#)
- ***uint8\_t RTC\_AlarmTypeDef::AlarmDateWeekDay*** Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of [\*\*RTC\\_WeekDay\\_Definitions\*\*](#)
- ***uint32\_t RTC\_AlarmTypeDef::Alarm*** Specifies the alarm . This parameter can be a value of [\*\*RTC\\_Alarms\\_Definitions\*\*](#)

## 32.1.5 RTC\_HandleTypeDef

**RTC\_HandleTypeDef** is defined in the stm32f0xx\_hal\_rtc.h

### Data Fields

- ***RTC\_TypeDef \* Instance***
- ***RTC\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_RTCStateTypeDef State***

### Field Documentation

- ***RTC\_TypeDef\* RTC\_HandleTypeDef::Instance*** Register base address
- ***RTC\_InitTypeDef RTC\_HandleTypeDef::Init*** RTC required parameters
- ***HAL\_LockTypeDef RTC\_HandleTypeDef::Lock*** RTC locking object
- ***\_\_IO HAL\_RTCStateTypeDef RTC\_HandleTypeDef::State*** Time communication state

## 32.2 RTC Firmware driver API description

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

### 32.2.1 RTC 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 to operate even when the main digital supply (VDD) is turned off, the VBAT pin powers the following blocks:

1. The RTC
2. The LSE oscillator
3. PC13 to PC15 I/Os (when available)

When the backup domain is supplied by VDD (analog switch connected to VDD), the following functions are available:

1. PC14 and PC15 can be used as either GPIO or LSE pins
2. PC13 can be used as a GPIO or as the RTC\_OUT pin

When the backup domain is supplied by VBAT (analog switch connected to VBAT because VDD is not present), the following functions are available:

1. PC14 and PC15 can be used as LSE pins only
2. PC13 can be used as the RTC\_OUT pin

### 32.2.2 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.

### 32.2.3 Backup Domain Access

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

1. Enable the Power Controller (PWR) APB1 interface clock using the \_\_PWR\_CLK\_ENABLE() function.
2. Enable access to RTC domain using the HAL\_PWR\_EnableBkUpAccess() function.
3. Select the RTC clock source using the \_\_HAL\_RCC\_RTC\_CONFIG() function.
4. Enable RTC Clock using the \_\_HAL\_RCC\_RTC\_ENABLE() function.

### 32.2.4 How to use RTC Driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL\_RTC\_Init() function.

#### Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL\_RTC\_SetTime() and HAL\_RTC\_SetDate() functions.
- To read the RTC Calendar, use the HAL\_RTC\_GetTime() and HAL\_RTC\_GetDate() functions.

### Alarm configuration

- To configure the RTC Alarm use the HAL\_RTC\_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL\_RTC\_SetAlarm\_IT() function.
- To read the RTC Alarm, use the HAL\_RTC\_GetAlarm() function.

### 32.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarm (Alarm A), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and Standby modes is possible only when the RTC clock source is LSE or LSI.

### 32.2.6 Initialization and de-initialization functions

This section provide functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
  - A 7-bit asynchronous prescaler and A 15-bit synchronous prescaler.
  - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC\_WPR.
3. To Configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC\_TR and RTC\_DR shadow registers. The HAL\_RTC\_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).
  - [\*\*HAL\\_RTC\\_Init\(\)\*\*](#)
  - [\*\*HAL\\_RTC\\_DelInit\(\)\*\*](#)
  - [\*\*HAL\\_RTC\\_MspInit\(\)\*\*](#)
  - [\*\*HAL\\_RTC\\_MspDelInit\(\)\*\*](#)

### 32.2.7 RTC Time and Date functions

This section provide functions allowing to configure Time and Date features

- [\*HAL\\_RTC\\_SetTime\(\)\*](#)
- [\*HAL\\_RTC\\_GetTime\(\)\*](#)
- [\*HAL\\_RTC\\_SetDate\(\)\*](#)
- [\*HAL\\_RTC\\_GetDate\(\)\*](#)

### 32.2.8 RTC Alarm functions

This section provide functions allowing to configure Alarm feature

- [\*HAL\\_RTC\\_SetAlarm\(\)\*](#)
- [\*HAL\\_RTC\\_SetAlarm\\_IT\(\)\*](#)
- [\*HAL\\_RTC\\_DeactivateAlarm\(\)\*](#)
- [\*HAL\\_RTC\\_GetAlarm\(\)\*](#)
- [\*HAL\\_RTC\\_AlarmIRQHandler\(\)\*](#)
- [\*HAL\\_RTC\\_AlarmAEventCallback\(\)\*](#)
- [\*HAL\\_RTC\\_PollForAlarmAEvent\(\)\*](#)

### 32.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization
- [\*HAL\\_RTC\\_WaitForSyncro\(\)\*](#)

### 32.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state
- [\*HAL\\_RTC\\_GetState\(\)\*](#)

### 32.2.11 HAL\_RTC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_RTC_Init ( <a href="#"><i>RTC_HandleTypeDef</i></a> * hrtc)</b>
Function Description	Initializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 32.2.12 HAL\_RTC\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_RTC_DelInit ( <i>RTC_HandleTypeDef</i> * hrtc)</b>
Function Description	DeInitializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• This function doesn't reset the RTC Backup Data registers.</li></ul>

### 32.2.13 HAL\_RTC\_MspInit

Function Name	<b>void HAL_RTC_MspInit ( <i>RTC_HandleTypeDef</i> * hrtc)</b>
Function Description	Initializes the RTC MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 32.2.14 HAL\_RTC\_MspDelInit

Function Name	<b>void HAL_RTC_MspDelInit ( <i>RTC_HandleTypeDef</i> * hrtc)</b>
Function Description	DeInitializes the RTC MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 32.2.15 HAL\_RTC\_SetTime

Function Name	<code>HAL_StatusTypeDef HAL_RTC_SetTime (</code> <code>RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime,</code> <code>uint32_t Format)</code>
Function Description	Sets RTC current time.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</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: <ul style="list-style-type: none"> <li>– <b>Format_BIN</b> Binary data format</li> <li>– <b>Format_BCD</b> BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.16 HAL\_RTC\_GetTime

Function Name	<code>HAL_StatusTypeDef HAL_RTC_GetTime (</code> <code>RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime,</code> <code>uint32_t Format)</code>
Function Description	Gets RTC current time.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</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: <ul style="list-style-type: none"> <li>– <b>Format_BIN</b> Binary data format</li> <li>– <b>Format_BCD</b> BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers.</li> </ul>

### 32.2.17 HAL\_RTC\_SetDate

Function Name	<code>HAL_StatusTypeDef HAL_RTC_SetDate ( <i>RTC_HandleTypeDef</i>* hrtc, <i>RTC_DateTypeDef</i>* sDate, uint32_t Format)</code>
Function Description	Sets RTC current date.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</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: <ul style="list-style-type: none"> <li>- <b>Format_BIN</b> Binary data format</li> <li>- <b>Format_BCD</b> BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.18 HAL\_RTC\_GetDate

Function Name	<code>HAL_StatusTypeDef HAL_RTC_GetDate ( <i>RTC_HandleTypeDef</i>* hrtc, <i>RTC_DateTypeDef</i>* sDate, uint32_t Format)</code>
Function Description	Gets RTC current date.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</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: <ul style="list-style-type: none"> <li>- <b>Format_BIN</b> Binary data format</li> <li>- <b>Format_BCD</b> BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.19 HAL\_RTC\_SetAlarm

Function Name	<code>HAL_StatusTypeDef HAL_RTC_SetAlarm ( <i>RTC_HandleTypeDef</i>* hrtc, <i>RTC_AlarmTypeDef</i>* sAlarm, uint32_t Format)</code>
Function Description	Sets the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>sAlarm</b> : Pointer to Alarm structure</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>Format :</b> Specifies the format of the entered parameters. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b>Format_BIN</b> Binary data format</li> <li>– <b>Format_BCD</b> BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.20 HAL\_RTC\_SetAlarm\_IT

Function Name	<code>HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (   RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,   uint32_t Format)</code>
Function Description	Sets the specified RTC Alarm with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc :</b> RTC handle</li> <li>• <b>sAlarm :</b> Pointer to Alarm structure</li> <li>• <b>Format :</b> Specifies the format of the entered parameters. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b>Format_BIN</b> Binary data format</li> <li>– <b>Format_BCD</b> BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The Alarm register can only be written when the corresponding Alarm is disabled (Use the <code>HAL_RTC_DeactivateAlarm()</code>).</li> <li>• The <code>HAL_RTC_SetTime()</code> must be called before enabling the Alarm feature.</li> </ul>

### 32.2.21 HAL\_RTC\_DeactivateAlarm

Function Name	<code>HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (   RTC_HandleTypeDef * hrtc, uint32_t Alarm)</code>
Function Description	Deactivate the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc :</b> RTC handle</li> <li>• <b>Alarm :</b> Specifies the Alarm. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b>ALARM_A</b> AlarmA</li> </ul> </li> </ul>

---

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

### 32.2.22 HAL\_RTC\_GetAlarm

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetAlarm (</b> <i>RTC_HandleTypeDef</i> * hrtc, <i>RTC_AlarmTypeDef</i> * 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> : RTC handle</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: <ul style="list-style-type: none"> <li>- <b>ALARM_A</b> AlarmA</li> </ul> </li> <li>• <b>Format</b> : Specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>Format_BIN</b> Binary data format</li> <li>- <b>Format_BCD</b> BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 32.2.23 HAL\_RTC\_AlarmIRQHandler

Function Name	<b>void HAL_RTC_AlarmIRQHandler (</b> <i>RTC_HandleTypeDef</i> * hrtc)
Function Description	This function handles Alarm interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

Notes

### 32.2.24 HAL\_RTC\_AlarmAEventCallback

Function Name	<b>void HAL_RTC_AlarmAEventCallback ( <i>RTC_HandleTypeDef</i> * hrtc)</b>
Function Description	Alarm A callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.25 HAL\_RTC\_PollForAlarmAEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent ( <i>RTC_HandleTypeDef</i> * hrtc, uint32_t Timeout)</b>
Function Description	This function handles AlarmA Polling request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.26 HAL\_RTC\_WaitForSynchro

Function Name	<b>HAL_StatusTypeDef HAL_RTC_WaitForSynchro ( <i>RTC_HandleTypeDef</i> * hrtc)</b>
Function Description	Waits until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The RTC Resynchronization mode is write protected, use the <code>__HAL_RTC_WRITEPROTECTION_DISABLE()</code> before calling this function.</li> <li>• To read the calendar through the shadow registers after</li> </ul>

Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC\_TR and RTC\_DR shadow registers.

### 32.2.27 HAL\_RTC\_GetState

Function Name	<b>HAL_RTCStateTypeDef HAL_RTC_GetState (</b> <b>RTC_HandleTypeDef * hrtc)</b>
Function Description	Returns the Alarm state.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 32.3 RTC Firmware driver defines

### 32.3.1 RTC

RTC

***RTC AlarmDateWeekDay Definitions***

RTC\_ALARMDATEWEEKDAYSEL\_DATE

RTC\_ALARMDATEWEEKDAYSEL\_WEEKDAY

IS\_RTC\_ALARM\_DATE\_WEEKDAY\_SEL

***RTC AlarmMask Definitions***

RTC\_ALARMMASK\_NONE

RTC\_ALARMMASK\_DATEWEEKDAY

RTC\_ALARMMASK\_HOURS

RTC\_ALARMMASK\_MINUTES

RTC\_ALARMMASK\_SECONDS

RTC\_ALARMMASK\_ALL

IS\_ALARM\_MASK

***RTC Alarms Definitions***

**RTC\_ALARM\_A****IS\_ALARM*****RTC Alarm Definitions*****IS\_RTC\_ALARM\_DATE\_WEEKDAY\_DATE****IS\_RTC\_ALARM\_DATE\_WEEKDAY\_WEEKDAY*****RTC Alarm Sub Seconds Masks Definitions***

<b>RTC_ALARMSUBSECONDMASK_ALL</b>	All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm
<b>RTC_ALARMSUBSECONDMASK_SS14_1</b>	SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.
<b>RTC_ALARMSUBSECONDMASK_SS14_2</b>	SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_3</b>	SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_4</b>	SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_5</b>	SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_6</b>	SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_7</b>	SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_8</b>	SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_9</b>	SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_10</b>	SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_11</b>	SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_12</b>	SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14_13</b>	SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared
<b>RTC_ALARMSUBSECONDMASK_SS14</b>	SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared
<b>RTC_ALARMSUBSECONDMASK_None</b>	SS[14:0] are compared and must match to activate alarm.

**IS\_RTC\_ALARM\_SUB\_SECOND\_MASK*****RTC Alarm Sub Seconds Value*****IS\_RTC\_ALARM\_SUB\_SECOND\_VALUE*****RTC AM PM Definitions***

RTC\_HOURFORMAT12\_AM

RTC\_HOURFORMAT12\_PM

IS\_RTC\_HOURFORMAT12

***RTC Asynchronous Predivider***

IS\_RTC\_ASYNCH\_PREDIV

***RTC DayLightSaving Definitions***

RTC\_DAYLIGHTSAVING\_SUB1H

RTC\_DAYLIGHTSAVING\_ADD1H

RTC\_DAYLIGHTSAVING\_NONE

IS\_RTC\_DAYLIGHT\_SAVING

***RTC Exported Macros***

`__HAL_RTC_RESET_HANDLE_STA  
TE`

**Description:**

- Reset RTC handle state.

**Parameters:**

- `__HANDLE__`: RTC handle.

**Return value:**

- None:

**Description:**

- Disable the write protection for RTC registers.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

**Description:**

- Enable the write protection for RTC registers.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

**Description:**

- Enable the RTC ALARMA peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

`_HAL_RTC_ALARMA_DISABLE`

**Description:**

- Disable the RTC ALARMA peripheral.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None:

`_HAL_RTC_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:

`_HAL_RTC_ALARM_DISABLE_IT`

**Description:**

- Disable the RTC Alarm interrupt.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.
- `_INTERRUPT_`: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - `RTC_IT_ALRA`: Alarm A interrupt

**Return value:**

- None:

`_HAL_RTC_ALARM_GET_IT`

**Description:**

- Check whether the specified RTC Alarm interrupt has occurred or not.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.
- `_FLAG_`: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be:
  - `RTC_IT_ALRA`: Alarm A interrupt

**Return value:**

- None:

`_HAL_RTC_ALARM_GET_FLAG`

**Description:**

- Get the selected RTC Alarm's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_ALRAF`
  - `RTC_FLAG_ALRAWF`

**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:

`RTC EXTI_LINE_ALARM_EVENT`  
External interrupt line 17 Connected to the RTC Alarm event

`RTC EXTI_LINE_TAMPER_TIMESTAMP_EVENT`  
External interrupt line 19 Connected to the RTC Tamper and Time Stamp events

`RTC EXTI_LINE_WAKEUPTIMER_EVENT`  
External interrupt line 20 Connected to the RTC Wakeup event

**Description:**

- Enable the RTC Exti line.

**Parameters:**

- `__EXTILINE__`: specifies the RTC Exti sources to be enabled or disabled. This parameter can be:
  - `RTC_EXTI_LINE_ALARM_EVENT`
  - `RTC_EXTI_LINE_TAMPER_TIMESTAMP_EVENT`
  - `RTC_EXTI_LINE_WAKEUPTIMER_EVENT`

**Return value:**

- None:

`__HAL_RTC_ENABLE_IT`  
`__HAL_RTC_DISABLE_IT`

**Description:**

- Disable the RTC Exti line.

**Parameters:**

- `__EXTILINE__`: specifies the RTC Exti sources to be enabled or disabled. This parameter can be:
  - `RTC_EXTI_LINE_ALARM_EVENT`
  - `RTC_EXTI_LINE_TAMPER_TIMESTAMP_EVENT`
  - `RTC_EXTI_LINE_WAKEUPTIMER_EVENT`

**Return value:**

- None:

`__HAL_RTC_DISABLE_IT``__HAL_RTC_EXTI_GENERATE_SWIT`**Description:**

- Generates a Software interrupt on selected EXTI line.

**Parameters:**

- `__EXTILINE__`: specifies the RTC Exti sources to be enabled or disabled. This parameter can be:
  - `RTC_EXTI_LINE_ALARM_EVENT`
  - `RTC_EXTI_LINE_TAMPER_TIMESTAMP_EVENT`
  - `RTC_EXTI_LINE_WAKEUPTIMER_EVENT`

**Return value:**

- None:

`__HAL_RTC_EXTI_CLEAR_FLAG`**Description:**

- Clear the RTC Exti flags.

**Parameters:**

- `__FLAG__`: specifies the RTC Exti sources to be enabled or disabled. This parameter can be:
  - `RTC_EXTI_LINE_ALARM_EVENT`
  - `RTC_EXTI_LINE_TAMPER_TIMESTAMP_EVENT`
  - `RTC_EXTI_LINE_WAKEUPTIMER_EVENT`

**Return value:**

- None:

`__HAL_RTC_CLEAR_FLAG`***RTC Flags Definitions***`RTC_FLAG_RECALPF``RTC_FLAG_TAMP3F``RTC_FLAG_TAMP2F`

RTC\_FLAG\_TAMP1F

RTC\_FLAG\_TSOVF

RTC\_FLAG\_TSF

RTC\_FLAG\_WUTF

RTC\_FLAG\_ALRAF

RTC\_FLAG\_INITF

RTC\_FLAG\_RSF

RTC\_FLAG\_INITS

RTC\_FLAG\_SHPF

RTC\_FLAG\_WUTWF

RTC\_FLAG\_ALRAWF

***RTC Hour Formats***

RTC\_HOURFORMAT\_24

RTC\_HOURFORMAT\_12

IS\_RTC\_HOUR\_FORMAT

***RTC Input parameter format definitions***

FORMAT\_BIN

FORMAT\_BCD

IS\_RTC\_FORMAT

***RTC Interrupts Definitions***

RTC\_IT\_TS

RTC\_IT\_WUT

RTC\_IT\_ALRA

RTC\_IT\_TAMP

RTC\_IT\_TAMP1

RTC\_IT\_TAMP2

RTC\_IT\_TAMP3

***RTC Mask Definition***

RTC\_TR\_RESERVED\_MASK

RTC\_DR\_RESERVED\_MASK

RTC\_INIT\_MASK

RTC\_RSF\_MASK

RTC\_TIMEOUT\_VALUE

***RTC Month Date Definitions***

RTC\_MONTH\_JANUARY

RTC\_MONTH\_FEBRUARY

RTC\_MONTH\_MARCH  
RTC\_MONTH\_APRL  
RTC\_MONTH\_MAY  
RTC\_MONTH\_JUNE  
RTC\_MONTH\_JULY  
RTC\_MONTH\_AUGUST  
RTC\_MONTH\_SEPTMBER  
RTC\_MONTH\_OCTOBER  
RTC\_MONTH\_NOVEMBER  
RTC\_MONTH\_DECEMBER  
IS\_RTC\_MONTH  
IS\_RTC\_DATE

***RTC Output Polarity Definitions***

RTC\_OUTPUT\_POLARITY\_HIGH  
RTC\_OUTPUT\_POLARITY\_LOW  
IS\_RTC\_OUTPUT\_POL

***RTC Output Type ALARM OUT***

RTC\_OUTPUT\_TYPE\_OPENDRAIN  
RTC\_OUTPUT\_TYPE\_PUSH\_PULL  
IS\_RTC\_OUTPUT\_TYPE

***RTC StoreOperation Definitions***

RTC\_STOREOPERATION\_RESET  
RTC\_STOREOPERATION\_SET  
IS\_RTC\_STORE\_OPERATION

***RTC Synchronous Predivider***

IS\_RTC\_SYNCH\_PREDIV

***RTC Time Definitions***

IS\_RTC\_HOUR12  
IS\_RTC\_HOUR24  
IS\_RTC\_MINUTES  
IS\_RTC\_SECONDS

***RTC WeekDay Definitions***

RTC\_WEEKDAY\_MONDAY  
RTC\_WEEKDAY\_TUESDAY  
RTC\_WEEKDAY\_WEDNESDAY  
RTC\_WEEKDAY\_THURSDAY

RTC\_WEEKDAY\_FRIDAY  
RTC\_WEEKDAY\_SATURDAY  
RTC\_WEEKDAY\_SUNDAY  
IS\_RTC\_WEEKDAY  
***RTC Year Date Definitions***  
IS\_RTC\_YEAR

## 33 HAL RTC Extension Driver

### 33.1 RTCEEx Firmware driver registers structures

#### 33.1.1 RTC\_TamperTypeDef

*RTC\_TamperTypeDef* is defined in the `stm32f0xx_hal_rtc_ex.h`

##### Data Fields

- *uint32\_t Tamper*
- *uint32\_t Trigger*
- *uint32\_t Filter*
- *uint32\_t SamplingFrequency*
- *uint32\_t PrechargeDuration*
- *uint32\_t TamperPullUp*
- *uint32\_t TimeStampOnTamperDetection*

##### Field Documentation

- *uint32\_t RTC\_TamperTypeDef::Tamper* Specifies the Tamper Pin. This parameter can be a value of [\*RTCEEx\\_Tamper\\_Pins\\_Definitions\*](#)
- *uint32\_t RTC\_TamperTypeDef::Trigger* Specifies the Tamper Trigger. This parameter can be a value of [\*RTCEEx\\_Tamper\\_Trigger\\_Definitions\*](#)
- *uint32\_t RTC\_TamperTypeDef::Filter* Specifies the RTC Filter Tamper. This parameter can be a value of [\*RTCEEx\\_Tamper\\_Filter\\_Definitions\*](#)
- *uint32\_t RTC\_TamperTypeDef::SamplingFrequency* Specifies the sampling frequency. This parameter can be a value of [\*RTCEEx\\_Tamper\\_Sampling\\_Frequencies\\_Definitions\*](#)
- *uint32\_t RTC\_TamperTypeDef::PrechargeDuration* Specifies the Precharge Duration . This parameter can be a value of [\*RTCEEx\\_Tamper\\_Pin\\_Precharge\\_Duration\\_Definitions\*](#)
- *uint32\_t RTC\_TamperTypeDef::TamperPullUp* Specifies the Tamper PullUp . This parameter can be a value of [\*RTCEEx\\_Tamper\\_Pull\\_UP\\_Definitions\*](#)
- *uint32\_t RTC\_TamperTypeDef::TimeStampOnTamperDetection* Specifies the TimeStampOnTamperDetection. This parameter can be a value of [\*RTCEEx\\_Tamper\\_TimeStampOnTamperDetection\\_Definitions\*](#)

### 33.2 RTCEEx Firmware driver API description

The following section lists the various functions of the RTCEEx 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 and Synchronous) and RTC hour format using the `HAL_RTC_Init()` function.

## RTC Wakeup configuration



Not available on F030x6/x8/xC and F070x6/xB

## TimeStamp configuration

- Configure the RTC\_AF trigger and enables the RTC TimeStamp using the HAL\_RTCEx\_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL\_RTCEx\_SetTimeStamp\_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL\_RTCEx\_GetTimeStamp() function.

## Tamper configuration

- Enable the RTC Tamper and Configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, precharge or discharge and Pull-UP using the HAL\_RTCEx\_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL\_RTCEx\_SetTamper\_IT() function.

## Backup Data Registers configuration



Not available on F030x6/x8/xC and F070x6/xB

### 33.2.2 RTC TimeStamp and Tamper functions

This section provide functions allowing to configure TimeStamp feature

- [\*HAL\\_RTCEx\\_SetTimeStamp\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetTimeStamp\\_IT\(\)\*](#)
- [\*HAL\\_RTCEx\\_DeactivateTimeStamp\(\)\*](#)
- [\*HAL\\_RTCEx\\_GetTimeStamp\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetTamper\(\)\*](#)
- [\*HAL\\_RTCEx\\_SetTamper\\_IT\(\)\*](#)
- [\*HAL\\_RTCEx\\_DeactivateTamper\(\)\*](#)
- [\*HAL\\_RTCEx\\_TamperTimeStampIRQHandler\(\)\*](#)
- [\*HAL\\_RTCEx\\_TimeStampEventCallback\(\)\*](#)
- [\*HAL\\_RTCEx\\_Tamper1EventCallback\(\)\*](#)
- [\*HAL\\_RTCEx\\_Tamper2EventCallback\(\)\*](#)
- [\*HAL\\_RTCEx\\_Tamper3EventCallback\(\)\*](#)
- [\*HAL\\_RTCEx\\_PollForTimeStampEvent\(\)\*](#)
- [\*HAL\\_RTCEx\\_PollForTamper1Event\(\)\*](#)
- [\*HAL\\_RTCEx\\_PollForTamper2Event\(\)\*](#)
- [\*HAL\\_RTCEx\\_PollForTamper3Event\(\)\*](#)

### 33.2.3 RTC Wake-up functions

This section provide functions allowing to configure Wake-up feature

- `HAL_RTCEEx_SetWakeUpTimer()`
- `HAL_RTCEEx_SetWakeUpTimer_IT()`
- `HAL_RTCEEx_DeactivateWakeUpTimer()`
- `HAL_RTCEEx_GetWakeUpTimer()`
- `HAL_RTCEEx_WakeUpTimerIRQHandler()`
- `HAL_RTCEEx_WakeUpTimerEventCallback()`
- `HAL_RTCEEx_PollForWakeUpTimerEvent()`

### 33.2.4 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 Coarse calibration parameters.
- Deactivates the Coarse calibration parameters
- Sets the Smooth calibration parameters.
- Configures the Synchronization Shift Control Settings.
- Configures the Calibration Pinout (RTC\_CALIB) Selection (1Hz or 512Hz).
- Deactivates the Calibration Pinout (RTC\_CALIB) Selection (1Hz or 512Hz).
- Enables the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enables the Bypass Shadow feature.
- Disables the Bypass Shadow feature.
- `HAL_RTCEEx_BKUPWrite()`
- `HAL_RTCEEx_BKUPRead()`
- `HAL_RTCEEx_SetSmoothCalib()`
- `HAL_RTCEEx_SetSynchroShift()`
- `HAL_RTCEEx_SetCalibrationOutPut()`
- `HAL_RTCEEx_DeactivateCalibrationOutPut()`
- `HAL_RTCEEx_SetRefClock()`
- `HAL_RTCEEx_DeactivateRefClock()`
- `HAL_RTCEEx_EnableBypassShadow()`
- `HAL_RTCEEx_DisableBypassShadow()`

### 33.2.5 HAL\_RTCEEx\_SetTimeStamp

Function Name	<code>HAL_StatusTypeDef HAL_RTCEEx_SetTimeStamp (</code> <code>RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge,</code> <code>uint32_t RTC_TimeStampPin)</code>
Function Description	Sets TimeStamp.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>TimeStampEdge</b> : Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following:</li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b><i>TimeStampEdge_Rising</i></b> the Time stamp event occurs on the rising edge of the related pin.</li> <li>- <b><i>TimeStampEdge_Falling</i></b> the Time stamp event occurs on the falling edge of the related pin.</li> </ul>
• <b>RTC_TimeStampPin</b> :	specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
	<ul style="list-style-type: none"> <li>- <b><i>RTC_TIMESTAMPPIN_PC13</i></b> PC13 is selected as RTC TimeStamp Pin.</li> </ul>
Return values	• <b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>• This API must be called before enabling the TimeStamp feature.</li> </ul>

### 33.2.6 HAL\_RTCEx\_SetTimeStamp\_IT

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (</b> <b><i>RTC_HandleTypeDef</i> * hrtc, uint32_t TimeStampEdge,</b> <b>uint32_t RTC_TimeStampPin)</b>
Function Description	Sets TimeStamp with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>TimeStampEdge</b> : Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following: <ul style="list-style-type: none"> <li>- <b><i>TimeStampEdge_Rising</i></b> the Time stamp event occurs on the rising edge of the related pin.</li> <li>- <b><i>TimeStampEdge_Falling</i></b> the Time stamp event occurs on the falling edge of the related pin.</li> </ul> </li> <li>• <b>RTC_TimeStampPin</b> : Specifies the RTC TimeStamp Pin. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>RTC_TIMESTAMPPIN_PC13</i></b> PC13 is selected as RTC TimeStamp Pin.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API must be called before enabling the TimeStamp feature.</li> </ul>

### 33.2.7 HAL\_RTCEx\_DeactivateTimeStamp

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (</code> <code>RTC_HandleTypeDef * hrtc)</code>
Function Description	Deactivates TimeStamp.
Parameters	<ul style="list-style-type: none"> <li>• <code>hrtc</code> : RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.8 HAL\_RTCEx\_GetTimeStamp

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (</code> <code>RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef *</code> <code>sTimeStamp, RTC_DateTypeDef *</code> <code>sTimeStampDate, uint32_t</code> <code>Format)</code>
Function Description	Gets the RTC TimeStamp value.
Parameters	<ul style="list-style-type: none"> <li>• <code>hrtc</code> : RTC handle</li> <li>• <code>sTimeStamp</code> : Pointer to Time structure</li> <li>• <code>sTimeStampDate</code> : Pointer to Date structure</li> <li>• <code>Format</code> : specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <code>Format_BIN</code> Binary data format</li> <li>– <code>Format_BCD</code> BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.9 HAL\_RTCEx\_SetTamper

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTamper (</code> <code>RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef *</code> <code>sTamper)</code>
Function Description	Sets Tamper.
Parameters	<ul style="list-style-type: none"> <li>• <code>hrtc</code> : RTC handle</li> <li>• <code>sTamper</code> : Pointer to Tamper Structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• By calling this API we disable the tamper interrupt for all</li> </ul>

tampers.

### 33.2.10 HAL\_RTCEx\_SetTamper\_IT

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (</code> <code>RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)</code>
Function Description	Sets Tamper with interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <code>hrtc</code> : RTC handle</li> <li>• <code>sTamper</code> : Pointer to RTC Tamper.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• By calling this API we force the tamper interrupt for all tampers.</li> </ul>

### 33.2.11 HAL\_RTCEx\_DeactivateTamper

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (</code> <code>RTC_HandleTypeDef * hrtc, uint32_t Tamper)</code>
Function Description	Deactivates Tamper.
Parameters	<ul style="list-style-type: none"> <li>• <code>hrtc</code> : RTC handle</li> <li>• <code>Tamper</code> : Selected tamper pin. This parameter can be any combination of RTC_TAMPER_1, RTC_TAMPER_2 and RTC_TAMPER_3.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.12 HAL\_RTCEx\_TamperTimeStampIRQHandler

Function Name	<code>void HAL_RTCEx_TamperTimeStampIRQHandler (</code> <code>RTC_HandleTypeDef * hrtc)</code>
---------------	---------------------------------------------------------------------------------------------------

---

Function Description	This function handles TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.13 HAL\_RTCEx\_TimeStampEventCallback

Function Name	<b>void HAL_RTCEx_TimeStampEventCallback (</b> <i>RTC_HandleTypeDef</i> * hrtc)
Function Description	TimeStamp callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.14 HAL\_RTCEx\_Tamper1EventCallback

Function Name	<b>void HAL_RTCEx_Tamper1EventCallback (</b> <i>RTC_HandleTypeDef</i> * hrtc)
Function Description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.15 HAL\_RTCEx\_Tamper2EventCallback

Function Name	<b>void HAL_RTCEx_Tamper2EventCallback (</b> <i>RTC_HandleTypeDef</i> * hrtc)
---------------	----------------------------------------------------------------------------------

Function Description	Tamper 2 callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.16 HAL\_RTCEx\_Tamper3EventCallback

Function Name	<b>void HAL_RTCEx_Tamper3EventCallback (</b> <b>RTC_HandleTypeDef * hrtc)</b>
Function Description	Tamper 3 callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.17 HAL\_RTCEx\_PollForTimeStampEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (</b> <b>RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	This function handles TimeStamp polling request.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.18 HAL\_RTCEx\_PollForTamper1Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (</b>
---------------	----------------------------------------------------------

***RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)***

Function Description	This function handles Tamper1 Polling.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.19 HAL\_RTCEx\_PollForTamper2Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (</b> <b><i>RTC_HandleTypeDef * hrtc, uint32_t Timeout)</i></b>
Function Description	This function handles Tamper2 Polling.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.20 HAL\_RTCEx\_PollForTamper3Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (</b> <b><i>RTC_HandleTypeDef * hrtc, uint32_t Timeout)</i></b>
Function Description	This function handles Tamper3 Polling.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.21 HAL\_RTCEx\_SetWakeUpTimer

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (</code> <code>RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter,</code> <code>uint32_t WakeUpClock)</code>
Function Description	Sets wake up timer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>WakeUpCounter</b> : Wake up counter</li> <li>• <b>WakeUpClock</b> : Wake up clock</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.22 HAL\_RTCEx\_SetWakeUpTimer\_IT

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (</code> <code>RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter,</code> <code>uint32_t WakeUpClock)</code>
Function Description	Sets wake up timer with interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>WakeUpCounter</b> : wake up counter</li> <li>• <b>WakeUpClock</b> : wake up clock</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.23 HAL\_RTCEx\_DeactivateWakeUpTimer

Function Name	<code>uint32_t HAL_RTCEx_DeactivateWakeUpTimer (</code> <code>RTC_HandleTypeDef * hrtc)</code>
Function Description	Deactivates wake up timer counter.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.24 HAL\_RTCEx\_GetWakeUpTimer

Function Name	<code>uint32_t HAL_RTCEx_GetWakeUpTimer ( RTC_HandleTypeDef * hrtc)</code>
Function Description	Gets wake up timer counter.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>Counter value</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.25 HAL\_RTCEx\_WakeUpTimerIRQHandler

Function Name	<code>void HAL_RTCEx_WakeUpTimerIRQHandler ( RTC_HandleTypeDef * hrtc)</code>
Function Description	This function handles Wake Up Timer interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.26 HAL\_RTCEx\_WakeUpTimerEventCallback

Function Name	<code>void HAL_RTCEx_WakeUpTimerEventCallback ( RTC_HandleTypeDef * hrtc)</code>
Function Description	Wake Up Timer callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.27 HAL\_RTCEx\_PollForWakeUpTimerEvent

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</code>
Function Description	This function handles Wake Up Timer Polling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.28 HAL\_RTCEx\_BKUPWrite

Function Name	<code>void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)</code>
Function Description	Writes a data in a specified RTC Backup data register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>BackupRegister</b> : RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 4 to specify the register.</li> <li>• <b>Data</b> : Data to be written in the specified RTC Backup data register.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.29 HAL\_RTCEx\_BKUPRead

Function Name	<code>uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)</code>
Function Description	Reads data from the specified RTC Backup data Register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>BackupRegister</b> : RTC Backup data Register number. This</li> </ul>

parameter can be: RTC\_BKP\_DRx where x can be from 0 to 4 to specify the register.

- |               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| Return values | <ul style="list-style-type: none"> <li>• <b>Read value</b></li> </ul> |
| Notes         | <ul style="list-style-type: none"> <li>• None.</li> </ul>             |

### 33.2.30 HAL\_RTCEx\_SetSmoothCalib

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (</code> <code>RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod,</code> <code>uint32_t SmoothCalibPlusPulses, uint32_t</code> <code>SmoothCalibMinusPulsesValue)</code>
Function Description	Sets the Smooth calibration parameters.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>SmoothCalibPeriod</b> : Select the Smooth Calibration Period. This parameter can be can be one of the following values : <ul style="list-style-type: none"> <li>– <b>RTC_SMOOTHCALIB_PERIOD_32SEC</b> The smooth calibration period is 32s.</li> <li>– <b>RTC_SMOOTHCALIB_PERIOD_16SEC</b> The smooth calibration period is 16s.</li> <li>– <b>RTC_SMOOTHCALIB_PERIOD_8SEC</b> The smooth calibartion period is 8s.</li> </ul> </li> <li>• <b>SmoothCalibPlusPulses</b> : Select to Set or reset the CALP bit. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>RTC_SMOOTHCALIB_PLUSPULSES_SET</b> Add one RTCCLK puls every <math>2^{11}</math> pulses.</li> <li>– <b>RTC_SMOOTHCALIB_PLUSPULSES_RESET</b> No RTCCLK pulses are added.</li> </ul> </li> <li>• <b>SmoothCalibMinusPulsesValue</b> : Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue mut be equal to 0.</li> </ul>

### 33.2.31 HAL\_RTCEx\_SetSynchroShift

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (</b> <b>RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)</b>
Function Description	Configures the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>ShiftAdd1S</b> : Select to add or not 1 second to the time calendar. This parameter can be one of the following values : <ul style="list-style-type: none"> <li>– <b>RTC_SHIFTADD1S_SET</b> Add one second to the clock calendar.</li> <li>– <b>RTC_SHIFTADD1S_RESET</b> No effect.</li> </ul> </li> <li>• <b>ShiftSubFS</b> : Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When REFCKON is set, firmware must not write to Shift control register.</li> </ul>

### 33.2.32 HAL\_RTCEx\_SetCalibrationOutPut

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (</b> <b>RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)</b>
Function Description	Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> <li>• <b>CalibOutput</b> : : Select the Calibration output Selection . This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>RTC_CALIBOUTPUT_512HZ</b> A signal has a regular waveform at 512Hz.</li> <li>– <b>RTC_CALIBOUTPUT_1HZ</b> A signal has a regular waveform at 1Hz.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.33 HAL\_RTCEx\_DeactivateCalibrationOutPut

Function Name	<b>HAL_StatusTypeDef</b>
Page-Footer	DocID026525 Rev 1

**HAL\_RTCEEx\_DeactivateCalibrationOutPut (**  
***RTC\_HandleTypeDef \* hrtc***)

Function Description	Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

**33.2.34 HAL\_RTCEEx\_SetRefClock**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEEx_SetRefClock (</b> <b><i>RTC_HandleTypeDef * hrtc</i></b> )
Function Description	Enables the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

**33.2.35 HAL\_RTCEEx\_DeactivateRefClock**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEEx_DeactivateRefClock (</b> <b><i>RTC_HandleTypeDef * hrtc</i></b> )
Function Description	Disable the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : RTC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

**33.2.36 HAL\_RTCEEx\_EnableBypassShadow**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (</b> <b>RTC_HandleTypeDef * hrtc)</b>
Function Description	Enables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.</li> </ul>

### 33.2.37 HAL\_RTCEx\_DisableBypassShadow

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (</b> <b>RTC_HandleTypeDef * hrtc)</b>
Function Description	Disables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.</li> </ul>

## 33.3 RTCEEx Firmware driver defines

### 33.3.1 RTCEEx

RTCEEx

***RTCEEx Add 1 Second Parameter Definition***

RTC\_SHIFTADD1S\_RESET

RTC\_SHIFTADD1S\_SET

IS\_RTC\_SHIFT\_ADD1S

***RTCEEx Backup Registers Definition***

RTC\_BKP\_DR0

RTC\_BKP\_DR1

RTC\_BKP\_DR2

RTC\_BKP\_DR3

RTC\_BKP\_DR4

IS\_RTC\_BKP

**RTCEx Calib Output selection Definition**

RTC\_CALIBOUTPUT\_512HZ

RTC\_CALIBOUTPUT\_1HZ

IS\_RTC\_CALIB\_OUTPUT

**RTCEx Exported Macros**

`_HAL_RTC_WAKEUPTIMER_ENABLE`

**Description:**

- Enable the RTC WakeUp Timer peripheral.

**Parameters:**

- `_HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

**Description:**

- Enable the RTC TimeStamp peripheral.

**Parameters:**

- `_HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

**Description:**

- Disable the RTC WakeUp Timer peripheral.

**Parameters:**

- `_HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

**Description:**

- Disable the RTC TimeStamp peripheral.

**Parameters:**

- `_HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

**Description:**

- Enable the RTC calibration output.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

`__HAL_RTC_CALIBRATION_OUTPUT_DISABLE`

- Disable the calibration output.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

`__HAL_RTC_CLOCKREF_DETECTION_ENABLE`

- Enable the clock reference detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

`__HAL_RTC_CLOCKREF_DETECTION_DISABLE`

- Disable the clock reference detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None:

`__HAL_RTC_TIMESTAMP_ENABLE_IT`

- Enable the RTC TimeStamp interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:

- RTC\_IT\_TS: TimeStamp interrupt

**Return value:**

- None:

`_HAL_RTC_WAKEUPTIMER_ENABLE_IT`

**Description:**

- Enable the RTC WakeUpTimer interrupt.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.
- `_INTERRUPT_`: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
  - RTC\_IT\_WUT: WakeUpTimer A interrupt

**Return value:**

- None:

`_HAL_RTC_TIMESTAMP_DISABLE_IT`

**Description:**

- Disable the RTC TimeStamp interrupt.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.
- `_INTERRUPT_`: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
  - RTC\_IT\_TS: TimeStamp interrupt

**Return value:**

- None:

`_HAL_RTC_WAKEUPTIMER_DISABLE_IT`

**Description:**

- Disable the RTC WakeUpTimer interrupt.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.
- `_INTERRUPT_`: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:

- RTC\_IT\_WUT:  
WakeUpTimer A interrupt

**Return value:**

- None:

`_HAL_RTC_TAMPER_GET_IT`

**Description:**

- Check whether the specified RTC Tamper interrupt has occurred or not.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.
- `_FLAG_`: specifies the RTC Tamper interrupt sources to be enabled or disabled. This parameter can be:
  - RTC\_IT\_TAMP1

**Return value:**

- None:

`_HAL_RTC_WAKEUPTIMER_GET_IT`

**Description:**

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.
- `_FLAG_`: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
  - RTC\_IT\_WUT:  
WakeUpTimer A interrupt

**Return value:**

- None:

`_HAL_RTC_TIMESTAMP_GET_IT`

**Description:**

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.
- `_FLAG_`: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
  - RTC\_IT\_TS: TimeStamp

interrupt

**Return value:**

- None:

`__HAL_RTC_TIMESTAMP_GET_FLAG`

**Description:**

- Get the selected RTC TimeStamp's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC TimeStamp Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_TSF`
  - `RTC_FLAG_TSOVF`

**Return value:**

- None:

`__HAL_RTC_WAKEUPTIMER_GET_FLAG`

**Description:**

- Get the selected RTC WakeUpTimer's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC WakeUpTimer Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_WUTF`
  - `RTC_FLAG_WUTWF`

**Return value:**

- None:

`__HAL_RTC_TAMPER_GET_FLAG`

**Description:**

- Get the selected RTC Tamper's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_TAMP1F`

**Return value:**

- None:

[\\_\\_HAL\\_RTC\\_SHIFT\\_GET\\_FLAG](#)**Description:**

- Get the selected RTC shift operation's flag status.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the RTC handle.
- [\\_\\_FLAG\\_\\_](#): specifies the RTC shift operation Flag is pending or not. This parameter can be:
  - [RTC\\_FLAG\\_SHPF](#)

**Return value:**

- None:

[\\_\\_HAL\\_RTC\\_TIMESTAMP\\_CLEAR\\_FLAG](#)**Description:**

- Clear the RTC Time Stamp's pending flags.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the RTC handle.
- [\\_\\_FLAG\\_\\_](#): specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - [RTC\\_FLAG\\_TSF](#)

**Return value:**

- None:

[\\_\\_HAL\\_RTC\\_TAMPER\\_CLEAR\\_FLAG](#)**Description:**

- Clear the RTC Tamper's pending flags.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the RTC handle.
- [\\_\\_FLAG\\_\\_](#): specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  - [RTC\\_FLAG\\_TAMP1F](#)

**Return value:**

- None:

[\\_\\_HAL\\_RTC\\_WAKEUPTIMER\\_CLEAR\\_FLAG](#)**Description:**

- Clear the RTC Wake Up timer's pending flags.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the RTC handle.

- `__FLAG__`: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  - `RTC_FLAG_WUTF`

**Return value:**

- None:

***RTCEx Output Selection Definition***

`RTC_OUTPUT_DISABLE`  
`RTC_OUTPUT_ALARMA`  
`RTC_OUTPUT_WAKEUP`  
`IS_RTC_OUTPUT`

***RTCEx Smooth calib Minus pulses Definition***

`IS_RTC_SMOOTH_CALIB_MINUS`

***RTCEx Smooth calib period Definition***

<code>RTC_SMOOTHCALIB_PERIOD_32SEC</code>	If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else $2^{\text{exp}20}$ RTCCLK seconds
<code>RTC_SMOOTHCALIB_PERIOD_16SEC</code>	If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else $2^{\text{exp}19}$ RTCCLK seconds
<code>RTC_SMOOTHCALIB_PERIOD_8SEC</code>	If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else $2^{\text{exp}18}$ RTCCLK seconds

`IS_RTC_SMOOTH_CALIB_PERIOD`

***RTCEx Smooth calib Plus pulses Definition***

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

`IS_RTC_SMOOTH_CALIB_PLUS`

***RTCEx Subtract Fraction Of Second Value***

`IS_RTC_SHIFT_SUBFS`

***RTCEx Tamper Filter Definition***

<code>RTC_TAMPERFILTER_DISABLE</code>	Tamper filter is disabled
<code>RTC_TAMPERFILTER_2SAMPLE</code>	Tamper is activated after 2 consecutive samples at the active level
<code>RTC_TAMPERFILTER_4SAMPLE</code>	Tamper is activated after 4 consecutive samples at the active level
<code>RTC_TAMPERFILTER_8SAMPLE</code>	Tamper is activated after 8 consecutive samples at the active level.

`IS_TAMPER_FILTER`

***RTCEx Tamper Pins Definition***

`RTC_TAMPER_1`

`RTC_TAMPER_2`

`RTC_TAMPER_3`

`IS_TAMPER`

***RTCEx Tamper Pin Precharge Duration Definition***

<code>RTC_TAMPERPRECHARGEDURATION_1RTCCLK</code>	Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
--------------------------------------------------	-------------------------------------------------------------------

<code>RTC_TAMPERPRECHARGEDURATION_2RTCCLK</code>	Tamper pins are pre-charged before sampling during 2 RTCCLK cycles
--------------------------------------------------	--------------------------------------------------------------------

<code>RTC_TAMPERPRECHARGEDURATION_4RTCCLK</code>	Tamper pins are pre-charged before sampling during 4 RTCCLK cycles
--------------------------------------------------	--------------------------------------------------------------------

<code>RTC_TAMPERPRECHARGEDURATION_8RTCCLK</code>	Tamper pins are pre-charged before sampling during 8 RTCCLK cycles
--------------------------------------------------	--------------------------------------------------------------------

`IS_TAMPER_PRECHARGE_DURATION`

***RTCEx Tamper Pull UP Definition***

<code>RTC_TAMPER_PULLUP_ENABLE</code>	TimeStamp on Tamper Detection event saved
---------------------------------------	-------------------------------------------

<code>RTC_TAMPER_PULLUP_DISABLE</code>	TimeStamp on Tamper Detection event is not saved
----------------------------------------	--------------------------------------------------

`IS_TAMPER_PULLUP_STATE`

***RTCEx Tamper Sampling Frequencies Definition***

<code>RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768</code>	Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
-----------------------------------------------------	-------------------------------------------------------------------------

<code>RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384</code>	Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
-----------------------------------------------------	-------------------------------------------------------------------------

<code>RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192</code>	Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192
----------------------------------------------------	------------------------------------------------------------------------

<code>RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096</code>	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
----------------------------------------------------	------------------------------------------------------------------------

<code>RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048</code>	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
----------------------------------------------------	------------------------------------------------------------------------

<code>RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024</code>	Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
----------------------------------------------------	------------------------------------------------------------------------

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256
IS_TAMPER_SAMPLING_FREQ	
<b>RTCE Tamper TimeStampOnTamperDetection Definition</b>	
RTC_TIMESTAMPONTAMPERDETECTION_ENABLE	TimeStamp on Tamper Detection event saved
RTC_TIMESTAMPONTAMPERDETECTION_DISABLE	TimeStamp on Tamper Detection event is not saved
IS_TAMPER_TIMESTAMPONTAMPER_DETECTION	
<b>RTCE Tamper Trigger Definition</b>	
RTC_TAMPERTRIGGER_RISINGEDGE	
RTC_TAMPERTRIGGER_FALLINGEDGE	
RTC_TAMPERTRIGGER_LOWLEVEL	
RTC_TAMPERTRIGGER_HIGHLEVEL	
IS_TAMPER_TRIGGER	
<b>RTCE TimeStamp Pin Selection</b>	
RTC_TIMESTAMPPIN_PC13	
IS_RTC_TIMESTAMP_PIN	
<b>RTCE Time Stamp Edges definition</b>	
RTC_TIMESTAMPEDGE_RISING	
RTC_TIMESTAMPEDGE_FALLING	
IS_TIMESTAMP_EDGE	
<b>RTCE Wakeup Timer Definition</b>	
RTC_WAKEUPCLOCK_RTCCLK_DIV16	
RTC_WAKEUPCLOCK_RTCCLK_DIV8	
RTC_WAKEUPCLOCK_RTCCLK_DIV4	
RTC_WAKEUPCLOCK_RTCCLK_DIV2	
RTC_WAKEUPCLOCK_CK_SPRE_16BITS	
RTC_WAKEUPCLOCK_CK_SPRE_17BITS	
IS_WAKEUP_CLOCK	
IS_WAKEUP_COUNTER	

## 34 HAL SMARTCARD Generic Driver

### 34.1 SMARTCARD Firmware driver registers structures

#### 34.1.1 SMARTCARD\_InitTypeDef

*SMARTCARD\_InitTypeDef* is defined in the `stm32f0xx_hal_smartcard.h`

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint16\_t Parity*
- *uint16\_t Mode*
- *uint16\_t CLKPolarity*
- *uint16\_t CLKPhase*
- *uint16\_t CLKLastBit*
- *uint16\_t OneBitSampling*
- *uint8\_t Prescaler*
- *uint8\_t GuardTime*
- *uint16\_t NACKEnable*
- *uint32\_t TimeOutEnable*
- *uint32\_t TimeOutValue*
- *uint8\_t BlockLength*
- *uint8\_t AutoRetryCount*

##### Field Documentation

- *uint32\_t SMARTCARD\_InitTypeDef::BaudRate* Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hsmartcard->Init.BaudRate)))
- *uint32\_t SMARTCARD\_InitTypeDef::WordLength* Specifies the number of data bits transmitted or received in a frame. This parameter **SMARTCARD\_Word\_Length** can only be set to 9 (8 data + 1 parity bits).
- *uint32\_t SMARTCARD\_InitTypeDef::StopBits* Specifies the number of stop bits **SMARTCARD\_Stop\_Bits**. Only 1.5 stop bits are authorized in SmartCard mode.
- *uint16\_t SMARTCARD\_InitTypeDef::Parity* Specifies the parity mode. This parameter can be a value of **SMARTCARD\_Parity**  
**Note:**The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.
- *uint16\_t SMARTCARD\_InitTypeDef::Mode* Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of **SMARTCARD\_Mode**
- *uint16\_t SMARTCARD\_InitTypeDef::CLKPolarity* Specifies the steady state of the serial clock. This parameter can be a value of **SMARTCARD\_Clock\_Polarity**
- *uint16\_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**
- *uint16\_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**

- **`uint16_t SMARTCARD_InitTypeDef::OneBitSampling`** Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of **`SMARTCARD_OneBit_Sampling`**.
- **`uint8_t SMARTCARD_InitTypeDef::Prescaler`** Specifies the SmartCard Prescaler
- **`uint8_t SMARTCARD_InitTypeDef::GuardTime`** Specifies the SmartCard Guard Time
- **`uint16_t SMARTCARD_InitTypeDef::NACKEnable`** Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of **`SMARTCARD_NACK_State`**
- **`uint32_t SMARTCARD_InitTypeDef::TimeOutEnable`** Specifies whether the receiver timeout is enabled. This parameter can be a value of **`SMARTCARD_Timeout_Enable`**
- **`uint32_t SMARTCARD_InitTypeDef::TimeOutValue`** Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- **`uint8_t SMARTCARD_InitTypeDef::BlockLength`** Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- **`uint8_t SMARTCARD_InitTypeDef::AutoRetryCount`** Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)

### 34.1.2 SMARTCARD\_AdvFeatureInitTypeDef

**`SMARTCARD_AdvFeatureInitTypeDef`** is defined in the `stm32f0xx_hal_smartcard.h`

#### Data Fields

- **`uint32_t AdvFeatureInit`**
- **`uint32_t TxPinLevelInvert`**
- **`uint32_t RxPinLevelInvert`**
- **`uint32_t DataInvert`**
- **`uint32_t Swap`**
- **`uint32_t OverrunDisable`**
- **`uint32_t DMADisableonRxError`**
- **`uint32_t MSBFirst`**

#### Field Documentation

- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit`** Specifies which advanced SMARTCARD features is initialized. Several advanced features may be initialized at the same time. This parameter can be a value of **`SMARTCARD_Advanced_Features_Initialization_Type`**
- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert`** Specifies whether the TX pin active level is inverted. This parameter can be a value of **`SMARTCARD_Tx_Inv`**
- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert`** Specifies whether the RX pin active level is inverted. This parameter can be a value of **`SMARTCARD_Rx_Inv`**
- **`uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert`** Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of **`SMARTCARD_Data_Inv`**

- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap` Specifies whether TX and RX pins are swapped. This parameter can be a value of `SMARTCARD_Rx_Tx_Swap`
- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable` Specifies whether the reception overrun detection is disabled. This parameter can be a value of `SMARTCARD_Overrun_Disable`
- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError` Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of `SMARTCARD_DMA_Disable_on_Rx_Error`
- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst` Specifies whether MSB is sent first on UART line. This parameter can be a value of `SMARTCARD_MSB_First`

### 34.1.3 SMARTCARD\_HandleTypeDef

`SMARTCARD_HandleTypeDef` is defined in the `stm32f0xx_hal_smartcard.h`

#### Data Fields

- `USART_TypeDef * Instance`
- `SMARTCARD_InitTypeDef Init`
- `SMARTCARD_AdvFeatureInitTypeDef AdvancedInit`
- `uint8_t * pTxBuffPtr`
- `uint16_t TxXferSize`
- `uint16_t TxXferCount`
- `uint8_t * pRxBuffPtr`
- `uint16_t RxXferSize`
- `uint16_t RxXferCount`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `HAL_LockTypeDef Lock`
- `HAL_SMARTCARD_StateTypeDef State`
- `HAL_SMARTCARD_ErrorTypeDef ErrorCode`

#### Field Documentation

- `USART_TypeDef* SMARTCARD_HandleTypeDef::Instance`
- `SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init`
- `SMARTCARD_AdvFeatureInitTypeDef SMARTCARD_HandleTypeDef::AdvancedInit`
- `uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr`
- `uint16_t SMARTCARD_HandleTypeDef::TxXferSize`
- `uint16_t SMARTCARD_HandleTypeDef::TxXferCount`
- `uint8_t* SMARTCARD_HandleTypeDef::pRxBuffPtr`
- `uint16_t SMARTCARD_HandleTypeDef::RxXferSize`
- `uint16_t SMARTCARD_HandleTypeDef::RxXferCount`
- `DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmatx`
- `DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmarx`
- `HAL_LockTypeDef SMARTCARD_HandleTypeDef::Lock`
- `HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::State`
- `HAL_SMARTCARD_ErrorTypeDef SMARTCARD_HandleTypeDef::ErrorCode`

## 34.2 SMARTCARD Firmware driver API description

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

### 34.2.1 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:
  - Enable the USARTx interface clock.
  - SMARTCARD pins configuration:
    - Enable the clock for the SMARTCARD GPIOs.
    - Configure these SMARTCARD pins as alternate function pull-up.
    - 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.
    - 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.
3. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsmartcard Init structure.
4. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard AdvancedInit structure.
5. Initialize the SMARTCARD associated USART 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() 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.
6. Three operation modes are available within this driver :

#### Polling mode IO operation

- Send an amount of data in blocking mode using HAL\_SMARTCARD\_Transmit()
- Receive an amount of data in blocking mode using HAL\_SMARTCARD\_Receive()

### Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL\_SMARTCARD\_Transmit\_IT()
- At transmission end of transfer HAL\_SMARTCARD\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_SMARTCARD\_Receive\_IT()
- At reception end of transfer HAL\_SMARTCARD\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_RxCpltCallback
- In case of transfer Error, HAL\_SMARTCARD\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_ErrorCallback

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL\_SMARTCARD\_Transmit\_DMA()
- At transmission end of transfer HAL\_SMARTCARD\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_SMARTCARD\_Receive\_DMA()
- At reception end of transfer HAL\_SMARTCARD\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_RxCpltCallback
- In case of transfer Error, HAL\_SMARTCARD\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_ErrorCallback

### SMARTCARD HAL driver macros list

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

- \_\_HAL\_SMARTCARD\_ENABLE: Enable the SMARTCARD peripheral
- \_\_HAL\_SMARTCARD\_DISABLE: Disable the SMARTCARD peripheral
- \_\_HAL\_SMARTCARD\_GET\_FLAG : Check whether the specified SMARTCARD flag is set or not
- \_\_HAL\_SMARTCARD\_CLEAR\_FLAG : Clear the specified SMARTCARD pending flag
- \_\_HAL\_SMARTCARD\_ENABLE\_IT: Enable the specified SMARTCARD interrupt
- \_\_HAL\_SMARTCARD\_DISABLE\_IT: Disable the specified SMARTCARD interrupt



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

### 34.2.2 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:
  - Baud Rate
  - Parity: parity should be enabled, Frame Length is fixed to 8 bits plus parity: the USART frame format is given in the tables below.
  - Receiver/transmitter modes
  - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
  - Prescaler value
  - Guard bit time
  - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - Time out enabling (and if activated, timeout value)
  - Block length
  - Auto-retry counter

**Table 21: USART frame formats**

M bit	PCE bit	USART frame
1	1	SB   8 bit data   PB   STB

or

M1, M0 bits	PCE bit	USART frame
01	1	SB   8 bit data   PB   STB

The HAL\_SMARTCARD\_Init() API follow respectively the USART (a)synchronous configuration procedures (details for the procedures are available in reference manual).

- [\*\*HAL\\_SMARTCARD\\_Init\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARD\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARD\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARD\\_MspDelInit\(\)\*\*](#)

### 34.2.3 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, These 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 API's are :
  - HAL\_SMARTCARD\_Transmit()
  - HAL\_SMARTCARD\_Receive()
3. Non Blocking mode API's with Interrupt are :
  - HAL\_SMARTCARD\_Transmit\_IT()
  - HAL\_SMARTCARD\_Receive\_IT()
  - HAL\_SMARTCARD\_IRQHandler()
4. Non Blocking mode functions with DMA are :
  - HAL\_SMARTCARD\_Transmit\_DMA()
  - HAL\_SMARTCARD\_Receive\_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_SMARTCARD\_TxCpltCallback()
  - HAL\_SMARTCARD\_RxCpltCallback()
  - HAL\_SMARTCARD\_ErrorCallback()
  - ***HAL\_SMARTCARD\_Transmit()***
  - ***HAL\_SMARTCARD\_Receive()***
  - ***HAL\_SMARTCARD\_Transmit\_IT()***
  - ***HAL\_SMARTCARD\_Receive\_IT()***
  - ***HAL\_SMARTCARD\_Transmit\_DMA()***
  - ***HAL\_SMARTCARD\_Receive\_DMA()***
  - ***HAL\_SMARTCARD\_IRQHandler()***
  - ***HAL\_SMARTCARD\_TxCpltCallback()***
  - ***HAL\_SMARTCARD\_RxCpltCallback()***
  - ***HAL\_SMARTCARD\_ErrorCallback()***

#### 34.2.4 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.
- SMARTCARD\_SetConfig() API configures the SMARTCARD peripheral

- SMARTCARD\_AdvFeatureConfig() API optionally configures the SMARTCARD advanced features
- SMARTCARD\_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization
- ***HAL\_SMARTCARD\_GetState()***
- ***HAL\_SMARTCARD\_GetError()***

### 34.2.5 HAL\_SMARTCARD\_Init

Function Name	<b><i>HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsmartcard)</i></b>
Function Description	Initializes the SMARTCARD mode according to the specified parameters in the SMARTCARD_InitTypeDef and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b><i>hsmartcard</i></b> : SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b><i>HAL status</i></b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 34.2.6 HAL\_SMARTCARD\_DelInit

Function Name	<b><i>HAL_StatusTypeDef HAL_SMARTCARD_DelInit (SMARTCARD_HandleTypeDef * hsmartcard)</i></b>
Function Description	Deinitializes the SMARTCARD peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b><i>hsmartcard</i></b> : SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b><i>HAL status</i></b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 34.2.7 HAL\_SMARTCARD\_MspInit

Function Name	<b><i>void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)</i></b>
Function Description	SMARTCARD MSP Init.

Parameters	<ul style="list-style-type: none"><li>• <b>hsmartcard</b> : SMARTCARD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 34.2.8 HAL\_SMARTCARD\_MspDelInit

Function Name	<b>void HAL_SMARTCARD_MspDelInit (</b> <b>SMARTCARD_HandleTypeDef * hsmartcard)</b>
Function Description	SMARTCARD MSP DelInit.
Parameters	<ul style="list-style-type: none"><li>• <b>hsmartcard</b> : SMARTCARD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 34.2.9 HAL\_SMARTCARD\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit (</b> <b>SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData,</b> <b>uint16_t Size, uint32_t Timeout)</b>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hsmartcard</b> : SMARTCARD handle</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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 34.2.10 HAL\_SMARTCARD\_Receive

---

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)</code>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</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> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 34.2.11 HAL\_SMARTCARD\_Transmit\_IT

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)</code>
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</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>• <b>HAL status</b></li> </ul>

### 34.2.12 HAL\_SMARTCARD\_Receive\_IT

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)</code>
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</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>• <b>HAL status</b></li> </ul>

## Notes

- None.

### 34.2.13 HAL\_SMARTCARD\_Transmit\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (     <b>SMARTCARD_HandleTypeDef</b> * hsmartcard, uint8_t * pData,     uint16_t Size)</code>
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 34.2.14 HAL\_SMARTCARD\_Receive\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (     <b>SMARTCARD_HandleTypeDef</b> * hsmartcard, uint8_t * pData,     uint16_t Size)</code>
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)</li> </ul>

### 34.2.15 HAL\_SMARTCARD\_IRQHandler

---

Function Name	<b>void HAL_SMARTCARD_IRQHandler (</b> <b>SMARTCARD_HandleTypeDef * hsmartcard)</b>
Function Description	SMARTCARD interrupt requests handling.
Parameters	<ul style="list-style-type: none"><li>• <b>hsmartcard</b> : SMARTCARD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 34.2.16 HAL\_SMARTCARD\_TxCpltCallback

Function Name	<b>void HAL_SMARTCARD_TxCpltCallback (</b> <b>SMARTCARD_HandleTypeDef * hsmartcard)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hsmartcard</b> : SMARTCARD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 34.2.17 HAL\_SMARTCARD\_RxCpltCallback

Function Name	<b>void HAL_SMARTCARD_RxCpltCallback (</b> <b>SMARTCARD_HandleTypeDef * hsmartcard)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hsmartcard</b> : SMARTCARD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 34.2.18 HAL\_SMARTCARD\_ErrorCallback

---

Function Name	<b>void HAL_SMARTCARD_ErrorCallback (</b> <b>SMARTCARD_HandleTypeDef * hsmartcard)</b>
Function Description	SMARTCARD error callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 34.2.19 HAL\_SMARTCARD\_GetState

Function Name	<b>HAL_SMARTCARD_StateTypeDef</b> <b>HAL_SMARTCARD_GetState ( SMARTCARD_HandleTypeDef * hsmartcard)</b>
Function Description	return the SMARTCARD state
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 34.2.20 HAL\_SMARTCARD\_GetError

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

## 34.3 SMARTCARD Firmware driver defines

### 34.3.1 SMARTCARD

SMARTCARD

**SMARTCARD advanced feature initialization type**

SMARTCARD\_ADVFEATURE\_NO\_INIT

SMARTCARD\_ADVFEATURE\_TXINVERT\_INIT

SMARTCARD\_ADVFEATURE\_RXINVERT\_INIT

SMARTCARD\_ADVFEATURE\_DATAINVERT\_INIT

SMARTCARD\_ADVFEATURE\_SWAP\_INIT

SMARTCARD\_ADVFEATURE\_RXOVERRUNDISABLE\_INIT

SMARTCARD\_ADVFEATURE\_DMADISABLEONERROR\_INIT

SMARTCARD\_ADVFEATURE\_MSBFIRST\_INIT

IS\_SMARTCARD\_ADVFEATURE\_INIT

**SMARTCARD Clock Phase**

SMARTCARD\_PHASE\_1EDGE

SMARTCARD\_PHASE\_2EDGE

IS\_SMARTCARD\_PHASE

**SMARTCARD Clock Polarity**

SMARTCARD\_POLARITY\_LOW

SMARTCARD\_POLARITY\_HIGH

IS\_SMARTCARD\_POLARITY

**SMARTCARD auto retry counter LSB position in CR3 register**

SMARTCARD\_CR3\_SCARCNT\_LSB\_POS

**SMARTCARD advanced feature Binary Data inversion**

SMARTCARD\_ADVFEATURE\_DATAINV\_DISABLE

SMARTCARD\_ADVFEATURE\_DATAINV\_ENABLE

IS\_SMARTCARD\_ADVFEATURE\_DATAINV

**SMARTCARD advanced feature DMA Disable on Rx Error**

SMARTCARD\_ADVFEATURE\_DMA\_ENABLEONRXERROR

SMARTCARD\_ADVFEATURE\_DMA\_DISABLEONRXERROR

IS\_SMARTCARD\_ADVFEATURE\_DMAONRXERROR

**SMARTCARD Exported Macros**

<code>_HAL_SMARTCARD_RESET_HANDLE_STATE</code>	<b>Description:</b>
	<ul style="list-style-type: none"><li>• Reset SMARTCARD handle state.</li></ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"><li>• <code>_HANDLE_</code>: SMARTCARD handle.</li></ul>

**Return value:**

- None:

**\_HAL\_SMARTCARD\_GET\_FLAG****Description:**

- Checks whether the specified Smartcard flag is set or not.

**Parameters:**

- \_HANDLE\_: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- \_FLAG\_: specifies the flag to check. This parameter can be one of the following values:
  - SMARTCARD\_FLAG\_RXACK: Receive enable acknowledge flag
  - SMARTCARD\_FLAG\_TEACK: Transmit enable acknowledge flag
  - SMARTCARD\_FLAG\_BUSY: Busy flag
  - SMARTCARD\_FLAG\_EOBF: End of block flag
  - SMARTCARD\_FLAG\_RTOF: Receiver timeout flag
  - 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\_ORE: OverRun Error flag
  - SMARTCARD\_FLAG\_NE: Noise Error flag
  - SMARTCARD\_FLAG\_FE: Framing Error flag
  - SMARTCARD\_FLAG\_PE: Parity Error flag

**Return value:**

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

**\_HAL\_SMARTCARD\_ENABLE\_IT****Description:**

- Enables the specified SmartCard interrupt.

**Parameters:**

- \_HANDLE\_: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- \_INTERRUPT\_: specifies the SMARTCARD interrupt to enable. This

parameter can be one of the following values:

- SMARTCARD\_IT\_EOBF: End Of Block interrupt
- SMARTCARD\_IT\_RTOF: Receive TimeOut interrupt
- 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\_PE: Parity Error interrupt
- SMARTCARD\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

[\\_\\_HAL\\_SMARTCARD\\_DISABLE\\_IT](#)

**Description:**

- Disables the specified SmartCard interrupt.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- [\\_\\_INTERRUPT\\_\\_](#): specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD\_IT\_EOBF: End Of Block interrupt
  - SMARTCARD\_IT\_RTOF: Receive TimeOut interrupt
  - 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\_PE: Parity Error interrupt
  - SMARTCARD\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

[\\_\\_HAL\\_SMARTCARD\\_GET\\_IT](#)

**Description:**

- Checks whether the specified SmartCard

interrupt has occurred or not.

#### Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- `__IT__`: specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
  - `SMARTCARD_IT_EOBF`: End Of Block interrupt
  - `SMARTCARD_IT_RTOF`: Receive TimeOut interrupt
  - `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_ORE`: OverRun Error interrupt
  - `SMARTCARD_IT_NE`: Noise Error interrupt
  - `SMARTCARD_IT_FE`: Framing Error interrupt
  - `SMARTCARD_IT_PE`: Parity Error interrupt

#### Return value:

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

### `__HAL_SMARTCARD_GET_IT_SOURCE`

#### Description:

- Checks whether the specified SmartCard interrupt source is enabled.

#### Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- `__IT__`: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - `SMARTCARD_IT_EOBF`: End Of Block interrupt
  - `SMARTCARD_IT_RTOF`: Receive TimeOut interrupt
  - `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\_ORE: OverRun Error interrupt
- SMARTCARD\_IT\_NE: Noise Error interrupt
- SMARTCARD\_IT\_FE: Framing Error interrupt
- SMARTCARD\_IT\_PE: Parity Error interrupt

**Return value:**

- The new state of IT (TRUE or FALSE).

[\\_\\_HAL\\_SMARTCARD\\_CLEAR\\_IT](#)**Description:**

- Clears the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

**Parameters:**

- HANDLE: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- IT\_CLEAR: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - USART\_CLEAR\_PEF: Parity Error Clear Flag
  - USART\_CLEAR\_FEF: Framing Error Clear Flag
  - USART\_CLEAR\_NEF: Noise detected Clear Flag
  - USART\_CLEAR\_OREF: OverRun Error Clear Flag
  - USART\_CLEAR\_TCF: Transmission Complete Clear Flag
  - USART\_CLEAR\_RTOF: Receiver Time Out Clear Flag
  - USART\_CLEAR\_EOBF: End Of Block Clear Flag

**Return value:**

- None:

[\\_\\_HAL\\_SMARTCARD\\_SEND\\_REQ](#)**Description:**

- Set a specific SMARTCARD request flag.

**Parameters:**

- HANDLE: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.

- REQ: specifies the request flag to set  
This parameter can be one of the following values:
  - SMARTCARD\_RXDATA\_FLUSH\_REQ: Receive Data flush Request
  - SMARTCARD\_TXDATA\_FLUSH\_REQ: Transmit data flush Request

**Return value:**

- None:

\_HAL\_SMARTCARD\_ENABLE

- Enable the USART associated to the SMARTCARD Handle.

**Parameters:**

- HANDLE: specifies the SMARTCARD Handle. The Handle Instance can be UARTx where x: 1, 2, 3 to select the USART peripheral

**Return value:**

- None:

\_HAL\_SMARTCARD\_DISABLE

- Disable the USART associated to the SMARTCARD Handle.

**Parameters:**

- HANDLE: specifies the SMARTCARD Handle. The Handle Instance can be UARTx where x: 1, 2, 3 to select the USART peripheral

**Return value:**

- None:

IS\_SMARTCARD\_BAUDRATE**Description:**

- Check the Baud rate range.

**Parameters:**

- BAUDRATE: Baud rate set by the configuration function.

**Return value:**

- Test: result (TRUE or FALSE)

IS\_SMARTCARD\_BLOCKLENGTH**Description:**

- Check the block length range.

**Parameters:**

- LENGTH: block length.

**Return value:**

**IS\_SMARTCARD\_TIMEOUT\_VALUE**

- Test: result (TRUE or FALSE)

**Description:**

- Check the receiver timeout value.

**Parameters:**

- \_\_TIMEOUTVALUE\_\_: receiver timeout value.

**Return value:**

- Test: result (TRUE or FALSE)

**IS\_SMARTCARD\_AUTORETRY\_CO**  
**NT****Description:**

- Check the SMARTCARD autoretry counter value.

**Parameters:**

- \_\_COUNT\_\_: number of retransmissions

**Return value:**

- Test: result (TRUE or FALSE)

***SMARTCARDEX Exported Macros*****\_HAL\_SMARTCARD\_GETCLOCKSOURCE****Description:**

- Reports the SMARTCARD clock source.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle
- \_\_CLOCKSOURCE\_\_: output variable

**Return value:**

- the: SMARTCARD clocking source, written in \_\_CLOCKSOURCE\_\_.

***SMARTCARD Flags***

SMARTCARD\_FLAG\_RXACK  
SMARTCARD\_FLAG\_TEACK  
SMARTCARD\_FLAG\_BUSY  
SMARTCARD\_FLAG\_EOBF  
SMARTCARD\_FLAG\_RTOF  
SMARTCARD\_FLAG\_TXE  
SMARTCARD\_FLAG\_TC  
SMARTCARD\_FLAG\_RXNE  
SMARTCARD\_FLAG\_ORE  
SMARTCARD\_FLAG\_NE

SMARTCARD\_FLAG\_FE

SMARTCARD\_FLAG\_PE

**SMARTCARD guard time value LSB position in GTPR register**

SMARTCARD\_GTPR\_GT\_LSB\_POS

**SMARTCARD interruptions flag mask**

SMARTCARD\_IT\_MASK

**SMARTCARD Interrupts Definition**

SMARTCARD\_IT\_PE

SMARTCARD\_IT\_TXE

SMARTCARD\_IT\_TC

SMARTCARD\_IT\_RXNE

SMARTCARD\_IT\_ERR

SMARTCARD\_IT\_ORE

SMARTCARD\_IT\_NE

SMARTCARD\_IT\_FE

SMARTCARD\_IT\_EOB

SMARTCARD\_IT\_RTO

**SMARTCARD Interruption Clear Flags**

SMARTCARD\_CLEAR\_PEF      Parity Error Clear Flag

SMARTCARD\_CLEAR\_FEF      Framing Error Clear Flag

SMARTCARD\_CLEAR\_NEF      Noise detected Clear Flag

SMARTCARD\_CLEAR\_OREF      OverRun Error Clear Flag

SMARTCARD\_CLEAR\_TCF      Transmission Complete Clear Flag

SMARTCARD\_CLEAR\_RTOF      Receiver Time Out Clear Flag

SMARTCARD\_CLEAR\_EOBF      End Of Block Clear Flag

**SMARTCARD Last Bit**

SMARTCARD\_LASTBIT\_DISABLED

SMARTCARD\_LASTBIT\_ENABLED

IS\_SMARTCARD\_LASTBIT

**SMARTCARD Transfer Mode**

SMARTCARD\_MODE\_RX

SMARTCARD\_MODE\_TX

SMARTCARD\_MODE\_TX\_RX

IS\_SMARTCARD\_MODE

**SMARTCARD advanced feature MSB first**

SMARTCARD\_ADVFEATURE\_MSBFIRST\_DISABLE

SMARTCARD\_ADVFEATURE\_MSBFIRST\_ENABLE

IS\_SMARTCARD\_ADVFEATURE\_MSBFIRST

**SMARTCARD NACK State**

SMARTCARD\_NACK\_ENABLED

SMARTCARD\_NACK\_DISABLED

IS\_SMARTCARD\_NACK

**SMARTCARD One Bit Sampling Method**

SMARTCARD\_ONEBIT\_SAMPLING\_DISABLED

SMARTCARD\_ONEBIT\_SAMPLING\_ENABLED

IS\_SMARTCARD\_ONEBIT\_SAMPLING

**SMARTCARD advanced feature Overrun Disable**

SMARTCARD\_ADVFEATURE\_OVERRUN\_ENABLE

SMARTCARD\_ADVFEATURE\_OVERRUN\_DISABLE

IS\_SMARTCARD\_OVERRUN

**SMARTCARD Parity**

SMARTCARD\_PARITY\_EVEN

SMARTCARD\_PARITY\_ODD

IS\_SMARTCARD\_PARITY

**SMARTCARD Private Constants**

TEACK\_REACK\_TIMEOUT

SMARTCARD\_TXDMA\_TIMEOUTVALUE

SMARTCARD\_TIMEOUT\_VALUE

USART\_CR1\_FIELDS

USART\_CR2\_CLK\_FIELDS

USART\_CR2\_FIELDS

USART\_CR3\_FIELDS

**SMARTCARD Request Parameters**

SMARTCARD\_RXDATA\_FLUSH\_REQUEST Receive Data flush Request

SMARTCARD\_TXDATA\_FLUSH\_REQUEST Transmit data flush Request

IS\_SMARTCARD\_REQUEST\_PARAMETER

**SMARTCARD block length LSB position in RTOR register**

SMARTCARD\_RTOR\_BLEN\_LSB\_POS

**SMARTCARD advanced feature RX pin active level inversion**

SMARTCARD\_ADVFEATURE\_RXINV\_DISABLE

SMARTCARD\_ADVFEATURE\_RXINV\_ENABLE

IS\_SMARTCARD\_ADVFEATURE\_RXINV

**SMARTCARD advanced feature RX TX pins swap**

SMARTCARD\_ADVFEATURE\_SWAP\_DISABLE

SMARTCARD\_ADVFEATURE\_SWAP\_ENABLE

IS\_SMARTCARD\_ADVFEATURE\_SWAP

***SMARTCARD Stop Bits***

SMARTCARD\_STOPBITS\_1\_5

IS\_SMARTCARD\_STOPBITS

***SMARTCARD Timeout Enable***

SMARTCARD\_TIMEOUT\_DISABLED

SMARTCARD\_TIMEOUT\_ENABLED

IS\_SMARTCARD\_TIMEOUT

***SMARTCARD advanced feature TX pin active level inversion***

SMARTCARD\_ADVFEATURE\_TXINV\_DISABLE

SMARTCARD\_ADVFEATURE\_TXINV\_ENABLE

IS\_SMARTCARD\_ADVFEATURE\_TXINV

***SMARTCARD Word Length***

SMARTCARD\_WORDLENGTH\_9B

IS\_SMARTCARD\_WORD\_LENGTH

## 35 HAL SMARTCARD Extension Driver

### 35.1 SMARTCARDEX Firmware driver API description

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

#### 35.1.1 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL\_SMARTCARDEX\_BlockLength\_Config() API allows to configure the Block Length on the fly
- HAL\_SMARTCARDEX\_TimeOut\_Config() API allows to configure the receiver timeout value on the fly
- HAL\_SMARTCARDEX\_EnableReceiverTimeOut() API enables the receiver timeout feature
- HAL\_SMARTCARDEX\_DisableReceiverTimeOut() API disables the receiver timeout feature
- [\*\*HAL\\_SMARTCARDEX\\_BlockLength\\_Config\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARDEX\\_TimeOut\\_Config\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARDEX\\_EnableReceiverTimeOut\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARDEX\\_DisableReceiverTimeOut\(\)\*\*](#)

#### 35.1.2 HAL\_SMARTCARDEX\_BlockLength\_Config

Function Name	<code>void HAL_SMARTCARDEX_BlockLength_Config (     <b>SMARTCARD_HandleTypeDef</b> * hsmartcard, uint8_t     BlockLength)</code>
Function Description	Update on the fly the SMARTCARD block length in RTOR register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</li> <li>• <b>BlockLength</b> : SMARTCARD block length (8-bit long at most)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 35.1.3 HAL\_SMARTCARDEX\_TimeOut\_Config

Function Name	<code>void HAL_SMARTCARDEX_TimeOut_Config (     <b>SMARTCARD_HandleTypeDef</b> * hsmartcard, uint32_t</code>
---------------	------------------------------------------------------------------------------------------------------------------

	<b>TimeOutValue)</b>
Function Description	Update on the fly the receiver timeout value in RTOR register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</li> <li>• <b>TimeOutValue</b> : receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0xFFFFFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 35.1.4 HAL\_SMARTCARDEX\_EnableReceiverTimeOut

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARDEX_EnableReceiverTimeOut (</b> <b>SMARTCARD_HandleTypeDef * hsmartcard)</b>
Function Description	Enable the SMARTCARD receiver timeout feature.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 35.1.5 HAL\_SMARTCARDEX\_DisableReceiverTimeOut

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARDEX_DisableReceiverTimeOut (</b> <b>SMARTCARD_HandleTypeDef * hsmartcard)</b>
Function Description	Disable the SMARTCARD receiver timeout feature.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmartcard</b> : SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 35.2 SMARTCARDEX Firmware driver defines

### 35.2.1 SMARTCARDEX

SMARTCARDEX

## 36 HAL SMBUS Generic Driver

### 36.1 SMBUS Firmware driver registers structures

#### 36.1.1 SMBUS\_InitTypeDef

**SMBUS\_InitTypeDef** is defined in the `stm32f0xx_hal_smbus.h`

##### Data Fields

- `uint32_t Timing`
- `uint32_t AnalogFilter`
- `uint32_t OwnAddress1`
- `uint32_t AddressingMode`
- `uint32_t DualAddressMode`
- `uint32_t OwnAddress2`
- `uint32_t OwnAddress2Masks`
- `uint32_t GeneralCallMode`
- `uint32_t NoStretchMode`
- `uint32_t PacketErrorCheckMode`
- `uint32_t PeripheralMode`
- `uint32_t SMBusTimeout`

##### Field Documentation

- `uint32_t SMBUS_InitTypeDef::Timing` Specifies the SMBUS\_TIMINGR\_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual
- `uint32_t SMBUS_InitTypeDef::AnalogFilter` Specifies if Analog Filter is enable or not. This parameter can be a value of [\*\*SMBUS\\_Analog\\_Filter\*\*](#)
- `uint32_t SMBUS_InitTypeDef::OwnAddress1` Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- `uint32_t SMBUS_InitTypeDef::AddressingMode` Specifies if 7-bit or 10-bit addressing mode for master is selected. This parameter can be a value of [\*\*SMBUS\\_addressing\\_mode\*\*](#)
- `uint32_t SMBUS_InitTypeDef::DualAddressMode` Specifies if dual addressing mode is selected. This parameter can be a value of [\*\*SMBUS\\_dual\\_addressing\\_mode\*\*](#)
- `uint32_t SMBUS_InitTypeDef::OwnAddress2` Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.
- `uint32_t SMBUS_InitTypeDef::OwnAddress2Masks` Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of [\*\*SMBUS\\_own\\_address2\\_masks\*\*](#).
- `uint32_t SMBUS_InitTypeDef::GeneralCallMode` Specifies if general call mode is selected. This parameter can be a value of [\*\*SMBUS\\_general\\_call\\_addressing\\_mode\*\*](#).
- `uint32_t SMBUS_InitTypeDef::NoStretchMode` Specifies if nostretch mode is selected. This parameter can be a value of [\*\*SMBUS\\_nostretch\\_mode\*\*](#)
- `uint32_t SMBUS_InitTypeDef::PacketErrorCheckMode` Specifies if Packet Error Check mode is selected. This parameter can be a value of [\*\*SMBUS\\_packet\\_error\\_check\\_mode\*\*](#)

- ***uint32\_t SMBUS\_InitTypeDef::PeripheralMode*** Specifies which mode of Peripheral is selected. This parameter can be a value of **SMBUS\_peripheral\_mode**
- ***uint32\_t SMBUS\_InitTypeDef::SMBusTimeout*** Specifies the content of the 32 Bits SMBUS\_TIMEOUT\_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

### 36.1.2 SMBUS\_HandleTypeDef

**SMBUS\_HandleTypeDef** is defined in the `stm32f0xx_hal_smbus.h`

#### Data Fields

- ***I2C\_TypeDef \* Instance***
- ***SMBUS\_InitTypeDef Init***
- ***uint8\_t \* pBuffPtr***
- ***uint16\_t XferSize***
- ***\_\_IO uint16\_t XferCount***
- ***\_\_IO uint32\_t XferOptions***
- ***\_\_IO HAL\_SMBUS\_StateTypeDef PreviousState***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_SMBUS\_StateTypeDef State***
- ***\_\_IO HAL\_SMBUS\_ErrorTypeDef ErrorCode***

#### Field Documentation

- ***I2C\_TypeDef\* SMBUS\_HandleTypeDef::Instance*** SMBUS registers base address
- ***SMBUS\_InitTypeDef SMBUS\_HandleTypeDef::Init*** SMBUS communication parameters
- ***uint8\_t\* SMBUS\_HandleTypeDef::pBuffPtr*** Pointer to SMBUS transfer buffer
- ***uint16\_t SMBUS\_HandleTypeDef::XferSize*** SMBUS transfer size
- ***\_\_IO uint16\_t SMBUS\_HandleTypeDef::XferCount*** SMBUS transfer counter
- ***\_\_IO uint32\_t SMBUS\_HandleTypeDef::XferOptions*** SMBUS transfer options
- ***\_\_IO HAL\_SMBUS\_StateTypeDef SMBUS\_HandleTypeDef::PreviousState*** SMBUS communication Previous state
- ***HAL\_LockTypeDef SMBUS\_HandleTypeDef::Lock*** SMBUS locking object
- ***\_\_IO HAL\_SMBUS\_StateTypeDef SMBUS\_HandleTypeDef::State*** SMBUS communication state
- ***\_\_IO HAL\_SMBUS\_ErrorTypeDef SMBUS\_HandleTypeDef::ErrorCode*** SMBUS Error code

## 36.2 SMBUS Firmware driver API description

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

### 36.2.1 Initialization and de-initialization functions

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

- User must Implement `HAL_SMBUS_MspInit()` function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC ).

- Call the function HAL\_SMBUS\_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Bus Timeout
  - Analog Filter mode
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
  - Packet Error Check mode
  - Peripheral mode
- Call the function HAL\_SMBUS\_DelInit() to restore the default configuration of the selected SMBUSx peripheral.
- [\*\*HAL\\_SMBUS\\_Init\(\)\*\*](#)
- [\*\*HAL\\_SMBUS\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_SMBUS\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_SMBUS\\_MspDelInit\(\)\*\*](#)

### 36.2.2 IO operation functions

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

1. Blocking mode function to check if device is ready for usage is :
  - HAL\_SMBUS\_IsDeviceReady()
2. There is only one mode of transfer:
  - No-Blocking mode : The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. No-Blocking mode functions with Interrupt are :
  - HAL\_SMBUS\_Master\_Transmit\_IT()
  - HAL\_SMBUS\_Master\_Receive\_IT()
  - HAL\_SMBUS\_Slave\_Transmit\_IT()
  - HAL\_SMBUS\_Slave\_Receive\_IT()
  - HAL\_SMBUS\_Slave\_Listen\_IT() or alias HAL\_SMBUS\_EnableListen\_IT()
  - HAL\_SMBUS\_DisableListen\_IT()
  - HAL\_SMBUS\_EnableAlert\_IT()
  - HAL\_SMBUS\_DisableAlert\_IT()
4. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
  - HAL\_SMBUS\_MasterTxCpltCallback()
  - HAL\_SMBUS\_MasterRxCpltCallback()
  - HAL\_SMBUS\_SlaveTxCpltCallback()
  - HAL\_SMBUS\_SlaveRxCpltCallback()
  - HAL\_SMBUS\_SlaveAddrCallback() or alias HAL\_SMBUS\_AddrCallback()
  - HAL\_SMBUS\_SlaveListenCpltCallback() or alias HAL\_SMBUS\_ListenCpltCallback()
  - HAL\_SMBUS\_ErrorCallback()
  - [\*\*HAL\\_SMBUS\\_Master\\_Transmit\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_SMBUS\\_Master\\_Receive\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_SMBUS\\_Master\\_Abort\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_SMBUS\\_Slave\\_Transmit\\_IT\(\)\*\*](#)

- `HAL_SMBUS_Slave_Receive_IT()`
- `HAL_SMBUS_Slave_Listen_IT()`
- `HAL_SMBUS_DisableListen_IT()`
- `HAL_SMBUS_EnableAlert_IT()`
- `HAL_SMBUS_DisableAlert_IT()`
- `HAL_SMBUS_IsDeviceReady()`
- `HAL_SMBUS_EV_IRQHandler()`
- `HAL_SMBUS_ER_IRQHandler()`
- `HAL_SMBUS_MasterTxCpltCallback()`
- `HAL_SMBUS_MasterRxCpltCallback()`
- `HAL_SMBUS_SlaveTxCpltCallback()`
- `HAL_SMBUS_SlaveRxCpltCallback()`
- `HAL_SMBUS_SlaveAddrCallback()`
- `HAL_SMBUS_SlaveListenCpltCallback()`
- `HAL_SMBUS_ErrorCallback()`

### 36.2.3 Peripheral State and Errors functions

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

- `HAL_SMBUS_GetState()`
- `HAL_SMBUS_GetError()`

### 36.2.4 HAL\_SMBUS\_Init

Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_Init (</code> <code>SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	Initializes the SMBUS according to the specified parameters in the <code>SMBUS_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : Pointer to a <code>SMBUS_HandleTypeDef</code> structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.5 HAL\_SMBUS\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_DelInit (</code> <code>SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	Deinitializes the SMBUS peripheral.

Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.6 HAL\_SMBUS\_MspInit

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

### 36.2.7 HAL\_SMBUS\_MspDeInit

Function Name	<b>void HAL_SMBUS_MspDeInit ( <i>SMBUS_HandleTypeDef</i> * hsmbus)</b>
Function Description	SMBUS MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.8 HAL\_SMBUS\_Master\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (</b> <b>SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress,</b> <b>uint8_t * pData, uint16_t Size, uint32_t XferOptions)</b>
Function Description	Transmit in master/host SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</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> <li>• <b>XferOptions</b> : Options of Transfer, value of SMBUS XferOptions definition</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.9 HAL\_SMBUS\_Master\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (</b> <b>SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress,</b> <b>uint8_t * pData, uint16_t Size, uint32_t XferOptions)</b>
Function Description	Receive in master/host SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</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> <li>• <b>XferOptions</b> : Options of Transfer, value of SMBUS XferOptions definition</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.10 HAL\_SMBUS\_Master\_Abort\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (</b> <b>SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)</b>
Function Description	Abort a master/host SMBUS process communication with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>DevAddress</b> : Target device address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• : This abort can be called only if state is ready</li> </ul>

### 36.2.11 HAL\_SMBUS\_Slave\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (</b> <b>SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)</b>
Function Description	Transmit in slave/device SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> <li>• <b>XferOptions</b> : Options of Transfer, value of SMBUS XferOptions definition</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.12 HAL\_SMBUS\_Slave\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (</b> <b>SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)</b>
Function Description	Receive in slave/device SMBUS mode an amount of data in no-

---

	blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> <li>• <b>XferOptions</b> : Options of Transfer, value of SMBUS XferOptions definition</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.13 HAL\_SMBUS\_Slave\_Listen\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_Slave_Listen_IT (</b> <b>SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	This function enable the Address listen mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.14 HAL\_SMBUS\_DisableListen\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (</b> <b>SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	This function disable the Address listen mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.15 HAL\_SMBUS\_EnableAlert\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (</b> <b>SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	This function enable the SMBUS alert mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.16 HAL\_SMBUS\_DisableAlert\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (</b> <b>SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	This function disable the SMBUS alert mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.17 HAL\_SMBUS\_IsDeviceReady

Function Name	<b>HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (</b> <b>SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress,</b> <b>uint32_t Trials, uint32_t Timeout)</b>
Function Description	Checks if target device is ready for communication.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>

	<ul style="list-style-type: none"><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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• This function is used with Memory devices</li></ul>

### 36.2.18 HAL\_SMBUS\_EV\_IRQHandler

Function Name	<b>void HAL_SMBUS_EV_IRQHandler ( <i>SMBUS_HandleTypeDef</i> * <i>hsmbus</i>)</b>
Function Description	This function handles SMBUS event interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hsmbus</b> : : Pointer to a <i>SMBUS_HandleTypeDef</i> structure that contains the configuration information for the specified SMBUS.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 36.2.19 HAL\_SMBUS\_ER\_IRQHandler

Function Name	<b>void HAL_SMBUS_ER_IRQHandler ( <i>SMBUS_HandleTypeDef</i> * <i>hsmbus</i>)</b>
Function Description	This function handles SMBUS error interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hsmbus</b> : : Pointer to a <i>SMBUS_HandleTypeDef</i> structure that contains the configuration information for the specified SMBUS.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 36.2.20 HAL\_SMBUS\_MasterTxCpltCallback

---

Function Name	<b>void HAL_SMBUS_MasterTxCpltCallback (</b> <b>SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Master Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.21 HAL\_SMBUS\_MasterRxCpltCallback

---

Function Name	<b>void HAL_SMBUS_MasterRxCpltCallback (</b> <b>SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Master Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.22 HAL\_SMBUS\_SlaveTxCpltCallback

---

Function Name	<b>void HAL_SMBUS_SlaveTxCpltCallback (</b> <b>SMBUS_HandleTypeDef * hsmbus)</b>
Function Description	Slave Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.23 HAL\_SMBUS\_SlaveRxCpltCallback

Function Name	<code>void HAL_SMBUS_SlaveRxCpltCallback (</code> <code>SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	Slave Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.24 HAL\_SMBUS\_SlaveAddrCallback

Function Name	<code>void HAL_SMBUS_SlaveAddrCallback (</code> <code>SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection,</code> <code>uint16_t AddrMatchCode)</code>
Function Description	Slave Address Match callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> <li>• <b>TransferDirection</b> : Master request Transfer Direction (Write/Read)</li> <li>• <b>AddrMatchCode</b> : Address Match Code</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.25 HAL\_SMBUS\_SlaveListenCpltCallback

Function Name	<code>void HAL_SMBUS_SlaveListenCpltCallback (</code> <code>SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	Listen Complete callbacks.

---

Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.26 HAL\_SMBUS\_ErrorCallback

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

### 36.2.27 HAL\_SMBUS\_GetState

Function Name	<b>HAL_SMBUS_StateTypeDef HAL_SMBUS_GetState ( <i>SMBUS_HandleTypeDef</i> * hsmbus)</b>
Function Description	Returns the SMBUS state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsmbus</b> : : SMBUS handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 36.2.28 HAL\_SMBUS\_GetError

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

## 36.3 SMBUS Firmware driver defines

### 36.3.1 SMBUS

SMBUS

***SMBUS addressing mode***

`SMBUS_ADDRESSINGMODE_7BIT`

`SMBUS_ADDRESSINGMODE_10BIT`

`IS_SMBUS_ADDRESSING_MODE`

***SMBUS Analog Filter***

`SMBUS_ANALOGFILTER_ENABLED`

`SMBUS_ANALOGFILTER_DISABLED`

`IS_SMBUS_ANALOG_FILTER`

***SMBUS dual addressing mode***

`SMBUS_DUALADDRESS_DISABLED`

`SMBUS_DUALADDRESS_ENABLED`

`IS_SMBUS_DUAL_ADDRESS`

***Input and Output operation functions***

`HAL_SMBUS_EnableListen_IT`

`HAL_SMBUS_AddrCallback`

`HAL_SMBUS_ListenCpltCallback`

***SMBUS Exported Macros***

`_HAL_SMBUS_RESET_HANDLE_STATE`   **Description:**

- Reset SMBUS handle state.

**Parameters:**

- `_HANDLE_`: SMBUS handle.

**Return value:**

- None:

`_HAL_SMBUS_ENABLE_IT`

**Description:**

- Enable or disable the specified SMBUS interrupts.

**Parameters:**

- `_HANDLE_`: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1 or 2 to select the SMBUS peripheral.
- `_INTERRUPT_`: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - `SMBUS_IT_ERRI`: Errors interrupt enable
  - `SMBUS_IT_TCI`: Transfer complete interrupt enable
  - `SMBUS_IT_STOPI`: STOP detection interrupt enable
  - `SMBUS_IT_NACKI`: NACK received interrupt enable
  - `SMBUS_IT_ADDRI`: Address match interrupt enable
  - `SMBUS_IT_RXI`: RX interrupt enable
  - `SMBUS_IT_TXI`: TX interrupt enable

**Return value:**

- None:

`_HAL_SMBUS_DISABLE_IT`

`_HAL_SMBUS_GET_IT_SOURCE`

**Description:**

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

**Parameters:**

- `_HANDLE_`: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1 or 2 to select the SMBUS peripheral.
- `_INTERRUPT_`: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  - `SMBUS_IT_ERRI`: Errors interrupt enable
  - `SMBUS_IT_TCI`: Transfer complete interrupt enable
  - `SMBUS_IT_STOPI`: STOP detection interrupt enable
  - `SMBUS_IT_NACKI`: NACK received interrupt enable

- received interrupt enable
- SMBUS\_IT\_ADDRI: Address match interrupt enable
- SMBUS\_IT\_RXI: RX interrupt enable
- SMBUS\_IT\_TXI: TX interrupt enable

**Return value:**

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

**SMBUS\_FLAG\_MASK****Description:**

- Checks whether the specified SMBUS flag is set or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1 or 2 to select the SMBUS peripheral.
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - SMBUS\_FLAG\_TXE: Transmit data register empty
  - SMBUS\_FLAG\_TXIS: Transmit interrupt status
  - SMBUS\_FLAG\_RXNE: Receive data register not empty
  - SMBUS\_FLAG\_ADDR: Address matched (slave mode)
  - SMBUS\_FLAG\_AF: NACK received flag
  - SMBUS\_FLAG\_STOPF: STOP detection flag
  - SMBUS\_FLAG\_TC: Transfer complete (master mode)
  - SMBUS\_FLAG\_TCR: Transfer complete reload
  - SMBUS\_FLAG\_BERR: Bus error
  - SMBUS\_FLAG\_ARLO: Arbitration lost
  - SMBUS\_FLAG\_OVR: Overrun/Underrun
  - SMBUS\_FLAG\_PECERR: PEC error in reception
  - SMBUS\_FLAG\_TIMEOUT: Timeout or Tlow detection flag
  - SMBUS\_FLAG\_ALERT: SMBus alert
  - SMBUS\_FLAG\_BUSY: Bus busy
  - SMBUS\_FLAG\_DIR: Transfer direction (slave mode)

**Return value:**

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

`_HAL_SMBUS_GET_FLAG`  
`_HAL_SMBUS_CLEAR_FLAG`

**Description:**

- Clears the SMBUS pending flags which are cleared by writing 1 in a specific bit.

**Parameters:**

- `_HANDLE_`: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1 or 2 to select the SMBUS peripheral.
- `_FLAG_`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `SMBUS_FLAG_ADDR`: Address matched (slave mode)
  - `SMBUS_FLAG_AF`: NACK received flag
  - `SMBUS_FLAG_STOPF`: STOP detection flag
  - `SMBUS_FLAG_BERR`: Bus error
  - `SMBUS_FLAG_ARLO`: Arbitration lost
  - `SMBUS_FLAG_OVR`: Overrun/Underrun
  - `SMBUS_FLAG_PECERR`: PEC error in reception
  - `SMBUS_FLAG_TIMEOUT`: Timeout or Tlow detection flag
  - `SMBUS_FLAG_ALERT`: SMBus alert

**Return value:**

- None:

`_HAL_SMBUS_ENABLE`  
`_HAL_SMBUS_DISABLE`  
`_HAL_SMBUS_RESET_CR1`  
`_HAL_SMBUS_RESET_CR2`  
`_HAL_SMBUS_GENERATE_START`  
`_HAL_SMBUS_GET_ADDR_MATCH`  
`_HAL_SMBUS_GET_DIR`  
`_HAL_SMBUS_GET_STOP_MODE`  
`_HAL_SMBUS_GET_PEC_MODE`  
`_HAL_SMBUS_GET_ALERT_ENABLED`

`__HAL_SMBUS_GENERATE_NACK`

`IS_SMBUS_OWN_ADDRESS1`

`IS_SMBUS_OWN_ADDRESS2`

***SMBUS Flag definition***

`SMBUS_FLAG_TXE`

`SMBUS_FLAG_TXIS`

`SMBUS_FLAG_RXNE`

`SMBUS_FLAG_ADDR`

`SMBUS_FLAG_AF`

`SMBUS_FLAG_STOPF`

`SMBUS_FLAG_TC`

`SMBUS_FLAG_TCR`

`SMBUS_FLAG_BERR`

`SMBUS_FLAG_ARLO`

`SMBUS_FLAG_OVR`

`SMBUS_FLAG_PECERR`

`SMBUS_FLAG_TIMEOUT`

`SMBUS_FLAG_ALERT`

`SMBUS_FLAG_BUSY`

`SMBUS_FLAG_DIR`

***SMBUS general call addressing mode***

`SMBUS_GENERALCALL_DISABLED`

`SMBUS_GENERALCALL_ENABLED`

`IS_SMBUS_GENERAL_CALL`

***SMBUS Interrupt configuration definition***

`SMBUS_IT_ERRI`

`SMBUS_IT_TCI`

`SMBUS_IT_STOPI`

`SMBUS_IT_NACKI`

`SMBUS_IT_ADDRI`

`SMBUS_IT_RXI`

`SMBUS_IT_TXI`

`SMBUS_IT_TX`

`SMBUS_IT_RX`

`SMBUS_IT_ALERT`

`SMBUS_IT_ADDR`

***SMBUS nostretch mode***

SMBUS\_NOSTRETCH\_DISABLED  
SMBUS\_NOSTRETCH\_ENABLED  
IS\_SMBUS\_NO\_STRETCH  
**SMBUS ownaddress2 masks**  
SMBUS\_OA2\_NOMASK  
SMBUS\_OA2\_MASK01  
SMBUS\_OA2\_MASK02  
SMBUS\_OA2\_MASK03  
SMBUS\_OA2\_MASK04  
SMBUS\_OA2\_MASK05  
SMBUS\_OA2\_MASK06  
SMBUS\_OA2\_MASK07  
IS\_SMBUS\_OWN\_ADDRESS2\_MASK  
**SMBUS packet error check mode**  
SMBUS\_PEC\_DISABLED  
SMBUS\_PEC\_ENABLED  
IS\_SMBUS\_PEC  
**SMBUS peripheral mode**  
SMBUS\_PERIPHERAL\_MODE\_SMBUS\_HOST  
SMBUS\_PERIPHERAL\_MODE\_SMBUS\_SLAVE  
SMBUS\_PERIPHERAL\_MODE\_SMBUS\_SLAVE\_ARP  
IS\_SMBUS\_PERIPHERAL\_MODE  
**SMBUS Private Define**  
TIMING\_CLEAR\_MASK  
HAL\_TIMEOUT\_ADDR  
HAL\_TIMEOUT\_BUSY  
HAL\_TIMEOUT\_DIR  
HAL\_TIMEOUT\_RXNE  
HAL\_TIMEOUT\_STOPF  
HAL\_TIMEOUT\_TC  
HAL\_TIMEOUT\_TCR  
HAL\_TIMEOUT\_TXIS  
MAX\_NBYTE\_SIZE  
**SMBUS Private Macros**  
\_\_SMBUS\_GET\_ISR\_REG  
\_\_SMBUS\_CHECK\_FLAG  
**SMBUS ReloadEndMode definition**

SMBUS\_SOFTEND\_MODE

SMBUS\_RELOAD\_MODE

SMBUS\_AUTOEND\_MODE

SMBUS\_SENDPEC\_MODE

IS\_SMBUS\_TRANSFER\_MODE

***SMBUS StartStopMode definition***

SMBUS\_NO\_STARTSTOP

SMBUS\_GENERATE\_STOP

SMBUS\_GENERATE\_START\_READ

SMBUS\_GENERATE\_START\_WRITE

IS\_SMBUS\_TRANSFER\_REQUEST

***SMBUS XferOptions definition***

SMBUS\_FIRST\_FRAME

SMBUS\_NEXT\_FRAME

SMBUS\_FIRST\_AND\_LAST\_FRAME\_NO\_PEC

SMBUS\_LAST\_FRAME\_NO\_PEC

SMBUS\_FIRST\_AND\_LAST\_FRAME\_WITH\_PEC

SMBUS\_LAST\_FRAME\_WITH\_PEC

IS\_SMBUS\_TRANSFER\_OPTIONS\_REQUEST

## 37 HAL SPI Generic Driver

### 37.1 SPI Firmware driver registers structures

#### 37.1.1 SPI\_InitTypeDef

*SPI\_InitTypeDef* is defined in the `stm32f0xx_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*
- *uint32\_t CRCLength*
- *uint32\_t NSSPMode*

##### Field Documentation

- *uint32\_t SPI\_InitTypeDef::Mode* Specifies the SPI operating mode. This parameter can be a value of [\*SPI\\_mode\*](#)
- *uint32\_t SPI\_InitTypeDef::Direction* Specifies the SPI bidirectional mode state. This parameter can be a value of [\*SPI\\_Direction\*](#)
- *uint32\_t SPI\_InitTypeDef::DataSize* Specifies the SPI data size. This parameter can be a value of [\*SPI\\_data\\_size\*](#)
- *uint32\_t SPI\_InitTypeDef::CLKPolarity* Specifies the serial clock steady state. This parameter can be a value of [\*SPI\\_Clock\\_Polarity\*](#)
- *uint32\_t SPI\_InitTypeDef::CLKPhase* Specifies the clock active edge for the bit capture. This parameter can be a value of [\*SPI\\_Clock\\_Phase\*](#)
- *uint32\_t SPI\_InitTypeDef::NSS* Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of [\*SPI\\_Slave\\_Select\\_management\*](#)
- *uint32\_t SPI\_InitTypeDef::BaudRatePrescaler* Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of [\*SPI\\_BaudRate\\_Prescaler\*](#)  
**Note:** The communication clock is derived from the master clock. The slave clock does not need to be set.
- *uint32\_t SPI\_InitTypeDef::FirstBit* Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of [\*SPI\\_MSB\\_LSB\\_transmission\*](#)
- *uint32\_t SPI\_InitTypeDef::TIMode* Specifies if the TI mode is enabled or not . This parameter can be a value of [\*SPI\\_TI\\_mode\*](#)
- *uint32\_t SPI\_InitTypeDef::CRCCalculation* Specifies if the CRC calculation is enabled or not. This parameter can be a value of [\*SPI\\_CRC\\_Calculation\*](#)

- ***uint32\_t SPI\_InitTypeDef::CRCPolynomial*** Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min\_Data = 0 and Max\_Data = 65535
- ***uint32\_t SPI\_InitTypeDef::CRCLength*** Specifies the CRC Length used for the CRC calculation. CRC Length is only used with Data8 and Data16, not other data size This parameter must 0 or 1 or 2
- ***uint32\_t SPI\_InitTypeDef::NSSPMode*** Specifies whether the NSSP signal is enabled or not . This parameter can be a value of ***SPI\_NSSP\_Mode*** This mode is activated by the NSSP bit in the SPIx\_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx\_CR1 CPHA = 0, CPOL setting is ignored)..

### 37.1.2 ***\_\_SPI\_HandleTypeDef***

***\_\_SPI\_HandleTypeDef*** is defined in the `stm32f0xx_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***
- ***uint32\_t CRCSize***
- ***void(\* RxISR***
- ***void(\* TxISR***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***HAL\_LockTypeDef Lock***
- ***HAL\_SPI\_StateTypeDef State***
- ***HAL\_SPI\_ErrorTypeDef ErrorCode***

#### Field Documentation

- ***SPI\_TypeDef\* \_\_SPI\_HandleTypeDef::Instance***
- ***SPI\_InitTypeDef \_\_SPI\_HandleTypeDef::Init***
- ***uint8\_t\* \_\_SPI\_HandleTypeDef::pTxBuffPtr***
- ***uint16\_t \_\_SPI\_HandleTypeDef::TxXferSize***
- ***uint16\_t \_\_SPI\_HandleTypeDef::TxXferCount***
- ***uint8\_t\* \_\_SPI\_HandleTypeDef::pRxBuffPtr***
- ***uint16\_t \_\_SPI\_HandleTypeDef::RxXferSize***
- ***uint16\_t \_\_SPI\_HandleTypeDef::RxXferCount***
- ***uint32\_t \_\_SPI\_HandleTypeDef::CRCSize***
- ***void(\* \_\_SPI\_HandleTypeDef::RxISR)(struct \_\_SPI\_HandleTypeDef \*hspi)***
- ***void(\* \_\_SPI\_HandleTypeDef::TxISR)(struct \_\_SPI\_HandleTypeDef \*hspi)***
- ***DMA\_HandleTypeDef\* \_\_SPI\_HandleTypeDef::hdmatx***
- ***DMA\_HandleTypeDef\* \_\_SPI\_HandleTypeDef::hdmarx***
- ***HAL\_LockTypeDef \_\_SPI\_HandleTypeDef::Lock***
- ***HAL\_SPI\_StateTypeDef \_\_SPI\_HandleTypeDef::State***
- ***HAL\_SPI\_ErrorTypeDef \_\_SPI\_HandleTypeDef::ErrorCode***

## 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 implement the HAL\_SPI\_MspInit ()API:
  - a. Enable the SPIx interface clock
  - b. SPI pins configuration
    - Enable the clock for the SPI GPIOs
    - Configure these SPI pins as alternate function push-pull
  - c. NVIC configuration if you need to use interrupt process
    - Configure the SPIx interrupt priority
    - Enable the NVIC SPI IRQ handle
  - d. DMA Configuration if you need to use DMA process
    - Declare a DMA\_HandleTypeDef handle structure for the transmit or receive channel
    - Enable the DMAx interface clock using
    - Configure the DMA handle parameters
    - Configure the DMA Tx or Rx channel
    - Associate the initialized hdma\_tx handle to the hspi DMA Tx or Rx handle
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel
3. Program the Mode, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL\_SPI\_Init() API:
  - These APIs configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL\_SPI\_MspInit(&hspi) API.

Using the HAL it is not possible to reach all supported SPI frequency with the different SPI Modes, the following table resume the max SPI frequency reached with 8- or 16-bit data size:

**Table 22: Maximum SPI frequency vs data size**

Process	Transfer mode	2 lines, fullduplex		1 line, Rx only		1 line	
Tx/Rx	Polling	$f_{CPU}/32$	$f_{CPU}/32$	NA	NA	NA	NA
	Interrupt	$f_{CPU}/32$	$f_{CPU}/32$	NA	NA	NA	NA
	DMA	$f_{CPU}/32$	$f_{CPU}/16$	NA	NA	NA	NA
Rx	Polling	$f_{CPU}/32$	$f_{CPU}/16$	$f_{CPU}/16$	$f_{CPU}/16$	$f_{CPU}/16$	$f_{CPU}/16$
	Interrupt	$f_{CPU}/16$	$f_{CPU}/16$	$f_{CPU}/16$	$f_{CPU}/16$	$f_{CPU}/16$	$f_{CPU}/16$
	DMA	$f_{CPU}/4$	$f_{CPU}/8$	$f_{CPU}/4$	$f_{CPU}/4$	$f_{CPU}/8$	$f_{CPU}/16$
Tx	Polling	$f_{CPU}/16$	$f_{CPU}/16$	NA	NA	$f_{CPU}/16$	$f_{CPU}/16$
	Interrupt	$f_{CPU}/32$	$f_{CPU}/16$	NA	NA	$f_{CPU}/16$	$f_{CPU}/16$
	DMA	$f_{CPU}/2$	$f_{CPU}/16$	NA	NA	$f_{CPU}/8$	$f_{CPU}/16$

@note The max SPI frequency depend on SPI data size (4bits, 5bits,..., 8bits,...15bits, 16bits), SPI mode(2 Lines fullduplex, 2 lines RxOnly, 1 line TX/RX) and Process mode (Polling, IT, DMA).

@note

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()

### 37.2.2 Initialization and Configuration functions

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

- User must Implement HAL\_SPI\_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function HAL\_SPI\_Init() to configure the selected device with the selected configuration:
  - Mode
  - Direction
  - Data Size
  - Clock Polarity and Phase
  - NSS Management
  - BaudRate Prescaler
  - FirstBit
  - TIMode
  - CRC Calculation
  - CRC Polynomial if CRC enabled
  - CRC Length, used only with Data8 and Data16
  - FIFO reception threshold
- Call the function HAL\_SPI\_DelInit() to restore the default configuration of the selected SPIx periperal.
- [\*\*HAL\\_SPI\\_Init\(\)\*\*](#)
- [\*\*HAL\\_SPI\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_SPI\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_SPI\\_MspDelInit\(\)\*\*](#)
- [\*\*HAL\\_SPI\\_InitExtended\(\)\*\*](#)

### 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.
  - 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 SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_SPI\_TxCpltCallback(), HAL\_SPI\_RxCpltCallback() and HAL\_SPI\_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process. The HAL\_SPI\_ErrorCallback() user callback will be executed when a communication error is detected.

2. Blocking mode APIs are :
  - HAL\_SPI\_Transmit() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_Receive() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_TransmitReceive() in full duplex mode
3. Non Blocking mode APIs with Interrupt are :
  - HAL\_SPI\_Transmit\_IT() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_Receive\_IT() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_TransmitReceive\_IT() in full duplex mode
  - HAL\_SPI\_IRQHandler()
4. Non Blocking mode functions with DMA are :
  - HAL\_SPI\_Transmit\_DMA() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_Receive\_DMA() in 1Line (simplex) and 2Lines (full duplex) mode
  - HAL\_SPI\_TransmitReceive\_DMA() in full duplex mode
5. A set of Transfer Complete Callbacks are provided in Non Blocking mode:
  - HAL\_SPI\_TxCpltCallback()
  - HAL\_SPI\_RxCpltCallback()
  - HAL\_SPI\_ErrorCallback()
  - HAL\_SPI\_TxRxCpltCallback()
  - [\*\*HAL\\_SPI\\_Transmit\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_Receive\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_TransmitReceive\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_Transmit\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_Receive\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_TransmitReceive\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_Transmit\\_DMA\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_Receive\\_DMA\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_TransmitReceive\\_DMA\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_DMAPause\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_DMAResume\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_DMAStop\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_IRQHandler\(\)\*\*](#)
  - [\*\*HAL\\_SPI\\_FlushRxFifo\(\)\*\*](#)
  - [\*\*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 Control 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	<code>HAL_StatusTypeDef HAL_SPI_Init ( SPI_HandleTypeDef * hspi)</code>
Function Description	Initializes the SPI according to the specified parameters in the <code>SPI_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"><li>• <code>hspi</code> : pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.6 HAL\_SPI\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_SPI_DeInit ( SPI_HandleTypeDef * hspi)</code>
Function Description	Deinitializes the SPI peripheral.
Parameters	<ul style="list-style-type: none"><li>• <code>hspi</code> : pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.7 HAL\_SPI\_MspInit

Function Name	<code>void HAL_SPI_MspInit ( SPI_HandleTypeDef * hspi)</code>
Function Description	SPI MSP Init.
Parameters	<ul style="list-style-type: none"><li>• <code>hspi</code> : pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None.</b></li></ul>

## Notes

- None.

### 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>• <b>None.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.9 HAL\_SPI\_InitExtended

Function Name	<b>HAL_StatusTypeDef HAL_SPI_InitExtended (</b> <b>SPI_HandleTypeDef * hspi)</b>
Function Description	
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.10 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.11 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.12 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.13 HAL\_SPI\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit_IT (</b> <b>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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 37.2.14 HAL\_SPI\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive_IT (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 37.2.15 HAL\_SPI\_TransmitReceive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</b>
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> </ul>

	<ul style="list-style-type: none"><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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.16 HAL\_SPI\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit_DMA (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.17 HAL\_SPI\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive_DMA (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.18 HAL\_SPI\_TransmitReceive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</b>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 37.2.19 HAL\_SPI\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DMAPause (</b> <b>SPI_HandleTypeDef * hspi)</b>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 37.2.20 HAL\_SPI\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DMAResume (</b> <b>SPI_HandleTypeDef * hspi)</b>
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</li> </ul>

module.

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

### 37.2.21 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.22 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 the specified SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>None.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.23 HAL\_SPI\_FlushRxFifo

Function Name	<b>HAL_StatusTypeDef HAL_SPI_FlushRxFifo (</b>
---------------	------------------------------------------------

**SPI\_HandleTypeDef \* hspi)**

Function Description	Flush the RX fifo.
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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.24 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>• <b>None.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.25 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>• <b>None.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.26 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>• <b>None.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.27 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>• <b>None.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.28 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>• <b>None.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.29 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>• <b>None.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.30 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>• <b>None.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.31 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>• <b>SPI state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 37.2.32 HAL\_SPI\_GetError

Function Name	<b>HAL_SPI_ErrorTypeDef HAL_SPI_GetError (</b> <b>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>• <b>SPI Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 37.3 SPI Firmware driver defines

### 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  
IS\_SPI\_BAUDRATE\_PRESCALER

***SPI Clock Phase***

SPI\_PHASE\_1EDGE  
SPI\_PHASE\_2EDGE  
IS\_SPI\_CPHA

***SPI Clock Polarity***

SPI\_POLARITY\_LOW  
SPI\_POLARITY\_HIGH  
IS\_SPI\_CPOL

***SPI CRC Calculation***

SPI\_CRCALCULATION\_DISABLED

SPI\_CRCALCULATION\_ENABLED

IS\_SPI\_CRC\_CALCULATION

***SPI CRC length***

SPI\_CRC\_LENGTH\_DATASIZE

SPI\_CRC\_LENGTH\_8BIT

SPI\_CRC\_LENGTH\_16BIT

IS\_SPI\_CRC\_LENGTH

***SPI Data size***

SPI\_DATASIZE\_4BIT

SPI\_DATASIZE\_5BIT

SPI\_DATASIZE\_6BIT

SPI\_DATASIZE\_7BIT

SPI\_DATASIZE\_8BIT

SPI\_DATASIZE\_9BIT

SPI\_DATASIZE\_10BIT

SPI\_DATASIZE\_11BIT

SPI\_DATASIZE\_12BIT

SPI\_DATASIZE\_13BIT

SPI\_DATASIZE\_14BIT

SPI\_DATASIZE\_15BIT

SPI\_DATASIZE\_16BIT

IS\_SPI\_DATASIZE

***SPI Direction***

SPI\_DIRECTION\_2LINES

SPI\_DIRECTION\_2LINES\_RXONLY

SPI\_DIRECTION\_1LINE

IS\_SPI\_DIRECTION

IS\_SPI\_DIRECTION\_2LINES

IS\_SPI\_DIRECTION\_2LINES\_OR\_1LINE

***SPI Exported Macros***

`_HAL_SPI_RESET_HANDLE_STATE` **Description:**

- Reset SPI handle state.

**Parameters:**

- `_HANDLE_`: SPI handle.

**Return value:**

- None:

[\\_\\_HAL\\_SPI\\_ENABLE\\_IT](#)**Description:**

- Enables or disables 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 or 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\\_DISABLE\\_IT](#)[\\_\\_HAL\\_SPI\\_GET\\_IT\\_SOURCE](#)**Description:**

- Checks 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:**

- Checks whether the specified SPI flag is set or not.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- [\\_\\_FLAG\\_\\_](#): specifies the flag to check. This parameter can be one of the following values:
  - [SPI\\_FLAG\\_RXNE](#): Receive buffer not

- empty flag
- SPI\_FLAG\_TXE: Transmit buffer empty flag
- SPI\_FLAG\_CRCERR: CRC error flag
- SPI\_FLAG\_MODF: Mode fault flag
- SPI\_FLAG\_OVR: Overrun flag
- SPI\_FLAG\_BSY: Busy flag
- SPI\_FLAG\_FRE: Frame format error flag
- SPI\_FLAG\_FTLVL: SPI fifo transmission level
- SPI\_FLAG\_FRLVL: SPI fifo reception level

**Return value:**

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

[\\_\\_HAL\\_SPI\\_CLEAR\\_CRCERRFLAG](#)**Description:**

- Clears 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:**

- Clears 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\\_OVRFAG](#)**Description:**

- Clears the SPI OVR pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None:

[\\_\\_HAL\\_SPI\\_CLEAR\\_FREFLAG](#)**Description:**

- Clears the SPI FRE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None:

`__HAL_SPI_ENABLE`

**Description:**

- 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:**

- None:

`__HAL_SPI_DISABLE`

**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:

`__HAL_SPI_1LINE_TX`

**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:

`__HAL_SPI_1LINE_RX`

**Description:**

- Sets the SPI receive-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:

`__HAL_SPI_RESET_CRC`

**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:

`IS_SPI_CRC_POLYNOMIAL`

***SPI FIFO reception threshold***

`SPI_RXFIFO_THRESHOLD`

`SPI_RXFIFO_THRESHOLD_QF`

`SPI_RXFIFO_THRESHOLD_HF`

***SPI Flag definition***

`SPI_FLAG_RXNE`

`SPI_FLAG_TXE`

`SPI_FLAG_BSY`

`SPI_FLAG_CRCERR`

`SPI_FLAG_MODF`

`SPI_FLAG_OVR`

`SPI_FLAG_FRE`

`SPI_FLAG_FTLVL`

`SPI_FLAG_FRLVL`

***SPI Interrupt configuration definition***

`SPI_IT_TXE`

`SPI_IT_RXNE`

`SPI_IT_ERR`

***SPI mode***

`SPI_MODE_SLAVE`

`SPI_MODE_MASTER`

`IS_SPI_MODE`

***SPI MSB LSB transmission***

`SPI_FIRSTBIT_MSB`

`SPI_FIRSTBIT_LSB`

`IS_SPI_FIRST_BIT`

***SPI NSS pulse management***

`SPI_NSS_PULSE_ENABLED`

`SPI_NSS_PULSE_DISABLED`

IS\_SPI\_NSSP

**SPI Private Constants**

SPI\_DEFAULT\_TIMEOUT

SPI\_FIFO\_SIZE

**SPI reception fifo status level**

SPI\_FRLVL\_EMPTY

SPI\_FRLVL\_QUARTER\_FULL

SPI\_FRLVL\_HALF\_FULL

SPI\_FRLVL\_FULL

**SPI Slave Select management**

SPI\_NSS\_SOFT

SPI\_NSS\_HARD\_INPUT

SPI\_NSS\_HARD\_OUTPUT

IS\_SPI\_NSS

**SPI TI mode**

SPI\_TIMODE\_DISABLED

SPI\_TIMODE\_ENABLED

IS\_SPI\_TIMODE

**SPI transmission fifo status level**

SPI\_FTLVL\_EMPTY

SPI\_FTLVL\_QUARTER\_FULL

SPI\_FTLVL\_HALF\_FULL

SPI\_FTLVL\_FULL

## 38 HAL TIM Generic Driver

### 38.1 TIM Firmware driver registers structures

#### 38.1.1 TIM\_Base\_InitTypeDef

*TIM\_Base\_InitTypeDef* is defined in the `stm32f0xx_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.

#### 38.1.2 TIM\_OC\_InitTypeDef

*TIM\_OC\_InitTypeDef* is defined in the `stm32f0xx_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.

### 38.1.3 TIM\_OnePulse\_InitTypeDef

***TIM\_OnePulse\_InitTypeDef*** is defined in the `stm32f0xx_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 IC\_Polarity***
- ***uint32\_t IC\_Selection***
- ***uint32\_t IC\_Filter***

#### Field Documentation

- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OCMode*** Specifies the TIM mode. This parameter can be a value of ***TIM\_Output\_Compare\_and\_PWM\_modes***
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::Pulse*** Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OCPolarity*** Specifies the output polarity. This parameter can be a value of ***TIM\_Output\_Compare\_Polarity***
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::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

**38.1.4 TIM\_IC\_InitTypeDef**

***TIM\_IC\_InitTypeDef*** is defined in the `stm32f0xx_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

**38.1.5 TIM\_Encoder\_InitTypeDef**

***TIM\_Encoder\_InitTypeDef*** is defined in the `stm32f0xx_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

### 38.1.6 TIM\_ClockConfigTypeDef

*TIM\_ClockConfigTypeDef* is defined in the `stm32f0xx_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 value of [TIM\\_Clock\\_Filter](#)

### 38.1.7 TIM\_ClearInputConfigTypeDef

*TIM\_ClearInputConfigTypeDef* is defined in the `stm32f0xx_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 value of *TIM\_ClearInput\_Filter*

**38.1.8 TIM\_SlaveConfigTypeDef**

*TIM\_SlaveConfigTypeDef* is defined in the *stm32f0xx\_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 value of *TIM\_Trigger\_Filter*

**38.1.9 TIM\_HandleTypeDef**

*TIM\_HandleTypeDef* is defined in the *stm32f0xx\_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

## 38.2 TIM Firmware driver API description

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

### 38.2.1 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

### 38.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
  - Time Base : `HAL_TIM_Base_MspInit()`
  - Input Capture : `HAL_TIM_IC_MspInit()`
  - Output Compare : `HAL_TIM_OC_MspInit()`
  - PWM generation : `HAL_TIM_PWM_MspInit()`
  - One-pulse mode output : `HAL_TIM_OnePulse_MspInit()`
  - Encoder mode output : `HAL_TIM_Encoder_MspInit()`
2. Initialize the TIM low level resources :
  - a. Enable the TIM interface clock using `__TIMx_CLK_ENABLE()`;
  - b. TIM pins configuration
    - Enable the clock for the TIM GPIOs using the following function: `__GPIOx_CLK_ENABLE()`;
    - Configure these TIM pins in Alternate function mode using `HAL_GPIO_Init()`;
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: `HAL_TIM_ConfigClockSource`, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
  - `HAL_TIM_Base_Init`: to use the Timer to generate a simple time base

- HAL\_TIM\_OC\_Init and HAL\_TIM\_OC\_ConfigChannel: to use the Timer to generate an Output Compare signal.
  - HAL\_TIM\_PWM\_Init and HAL\_TIM\_PWM\_ConfigChannel: to use the Timer to generate a PWM signal.
  - HAL\_TIM\_IC\_Init and HAL\_TIM\_IC\_ConfigChannel: to use the Timer to measure an external signal.
  - HAL\_TIM\_OnePulse\_Init and HAL\_TIM\_OnePulse\_ConfigChannel: to use the Timer in One Pulse Mode.
  - HAL\_TIM\_Encoder\_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
- Time Base : HAL\_TIM\_Base\_Start(), HAL\_TIM\_Base\_Start\_DMA(), HAL\_TIM\_Base\_Start\_IT()
  - Input Capture : HAL\_TIM\_IC\_Start(), HAL\_TIM\_IC\_Start\_DMA(), HAL\_TIM\_IC\_Start\_IT()
  - Output Compare : HAL\_TIM\_OC\_Start(), HAL\_TIM\_OC\_Start\_DMA(), HAL\_TIM\_OC\_Start\_IT()
  - PWM generation : HAL\_TIM\_PWM\_Start(), HAL\_TIM\_PWM\_Start\_DMA(), HAL\_TIM\_PWM\_Start\_IT()
  - One-pulse mode output : HAL\_TIM\_OnePulse\_Start(), HAL\_TIM\_OnePulse\_Start\_IT()
  - Encoder mode output : HAL\_TIM\_Encoder\_Start(), HAL\_TIM\_Encoder\_Start\_DMA(), HAL\_TIM\_Encoder\_Start\_IT().
6. The DMA Burst is managed with the two following functions:  
HAL\_TIM\_DMABurst\_WriteStart() HAL\_TIM\_DMABurst\_ReadStart()

### 38.2.3 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\(\)\*\*\*](#)

### 38.2.4 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\\_DelInit\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_MspDelInit\(\)\*](#)
- [\*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\(\)\*](#)

### 38.2.5 Time PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM OPWM.
- De-initialize the TIM PWM.
- Start the Time PWM.
- Stop the Time PWM.
- Start the Time PWM and enable interrupt.
- Stop the Time PWM and disable interrupt.
- Start the Time PWM and enable DMA transfer.
- Stop the Time PWM and disable DMA transfer.
- [\*HAL\\_TIM\\_PWM\\_Init\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_DelInit\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_MspDelInit\(\)\*](#)
- [\*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\(\)\*](#)

### 38.2.6 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\_DelInit\(\)`](#)
- [`HAL\_TIM\_IC\_MspInit\(\)`](#)
- [`HAL\_TIM\_IC\_MspDelInit\(\)`](#)
- [`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\(\)`](#)

### 38.2.7 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\_DelInit\(\)`](#)
- [`HAL\_TIM\_OnePulse\_MspInit\(\)`](#)
- [`HAL\_TIM\_OnePulse\_MspDelInit\(\)`](#)
- [`HAL\_TIM\_OnePulse\_Start\(\)`](#)
- [`HAL\_TIM\_OnePulse\_Stop\(\)`](#)
- [`HAL\_TIM\_OnePulse\_Start\_IT\(\)`](#)
- [`HAL\_TIM\_OnePulse\_Stop\_IT\(\)`](#)

### 38.2.8 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\_DelInit\(\)`](#)
- [`HAL\_TIM\_Encoder\_MspInit\(\)`](#)
- [`HAL\_TIM\_Encoder\_MspDelInit\(\)`](#)
- [`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\(\)\*](#)

### 38.2.9 IRQ handler management

This section provides Timer IRQ handler function.

- [\*HAL\\_TIM\\_IRQHandler\(\)\*](#)

### 38.2.10 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\(\)\*](#)

### 38.2.11 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\(\)\*](#)

### 38.2.12 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\(\)`](#)

### 38.2.13 HAL\_TIM\_Base\_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Base_Init ( TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM Time base Unit according to the specified parameters in the <code>TIM_HandleTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <code>htim</code> : : TIM Base handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.14 HAL\_TIM\_Base\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Base_DelInit ( TIM_HandleTypeDef * htim)</code>
Function Description	DeInitializes the TIM Base peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <code>htim</code> : : TIM Base handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.15 HAL\_TIM\_Base\_MspInit

Function Name	<b>void HAL_TIM_Base_MspInit ( <i>TIM_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.16 HAL\_TIM\_Base\_MspInit

Function Name	<b>void HAL_TIM_Base_MspInit ( <i>TIM_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.17 HAL\_TIM\_Base\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Starts the TIM Base generation.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.18 HAL\_TIM\_Base\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop ( <i>TIM_HandleTypeDef</i> * htim)</b>
---------------	-------------------------------------------------------------------------------

Function Description	Stops the TIM Base generation.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.19 HAL\_TIM\_Base\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Starts the TIM Base generation 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.20 HAL\_TIM\_Base\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Stops the TIM Base generation 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.21 HAL\_TIM\_Base\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t * pData, uint16_t Length)</b>
---------------	---------------------------------------------------------------------------------------------------------------------------------

---

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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.22 HAL\_TIM\_Base\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (</b> <i>TIM_HandleTypeDef</i> * <b>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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.23 HAL\_TIM\_OC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Init (</b> <i>TIM_HandleTypeDef</i> * <b>htim)</b>
Function Description	Initializes the TIM Output Compare according to the specified parameters in the <i>TIM_HandleTypeDef</i> 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.24 HAL\_TIM\_OC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_DeInit (</b> <b><i>TIM_HandleTypeDef</i> * 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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.25 HAL\_TIM\_OC\_MspInit

Function Name	<b>void HAL_TIM_OC_MspInit (</b> <b><i>TIM_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.26 HAL\_TIM\_OC\_MspDeInit

Function Name	<b>void HAL_TIM_OC_MspDeInit (</b> <b><i>TIM_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.27 HAL\_TIM\_OC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, uint32_t <i>Channel</i>)</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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.28 HAL\_TIM\_OC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, uint32_t <i>Channel</i>)</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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.29 HAL\_TIM\_OC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start_IT ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, uint32_t <i>Channel</i>)</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:           <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> TIM Channel 2 selected</li> <li>– <b>TIM_CHANNEL_3</b> TIM Channel 3 selected</li> <li>– <b>TIM_CHANNEL_4</b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<b>HAL status</b>
Notes	None.

### 38.2.30 HAL\_TIM\_OC\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Output Compare signal generation in interrupt mode.
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 be one of the following values:           <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> TIM Channel 2 selected</li> <li>– <b>TIM_CHANNEL_3</b> TIM Channel 3 selected</li> <li>– <b>TIM_CHANNEL_4</b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<b>HAL status</b>
Notes	None.

### 38.2.31 HAL\_TIM\_OC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</b>
Function Description	Starts the TIM Output Compare signal generation in DMA mode.
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:           <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul>
• <b>pData</b> :	The source Buffer address.
• <b>Length</b> :	The length of data to be transferred from memory to TIM peripheral
Return values	• <b>HAL status</b>
Notes	• None.

### 38.2.32 HAL\_TIM\_OC\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Output Compare signal generation in DMA mode.
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 be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 38.2.33 HAL\_TIM\_PWM\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Init (</b> <b><i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Initializes the TIM PWM Time Base according to the specified parameters in the <i>TIM_HandleTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 38.2.34 HAL\_TIM\_PWM\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_DeInit (</code> <i><code>TIM_HandleTypeDef * htim)</code></i>
Function Description	DeInitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"><li>• <code>htim</code> : : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.35 HAL\_TIM\_PWM\_MspInit

Function Name	<code>void HAL_TIM_PWM_MspInit (</code> <i><code>TIM_HandleTypeDef * htim)</code></i>
Function Description	Initializes the TIM PWM MSP.
Parameters	<ul style="list-style-type: none"><li>• <code>htim</code> : : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.36 HAL\_TIM\_PWM\_MspDeInit

Function Name	<code>void HAL_TIM_PWM_MspDeInit (</code> <i><code>TIM_HandleTypeDef * htim)</code></i>
Function Description	DeInitializes TIM PWM MSP.
Parameters	<ul style="list-style-type: none"><li>• <code>htim</code> : : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.37 HAL\_TIM\_PWM\_Start

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_Start (     <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</code>
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:<ul style="list-style-type: none"><li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li><li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li><li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li><li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.38 HAL\_TIM\_PWM\_Stop

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_Stop (     <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</code>
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:<ul style="list-style-type: none"><li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li><li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li><li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li><li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.39 HAL\_TIM\_PWM\_Start\_IT

---

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (</b> <b><i>TIM_HandleTypeDef</i> * 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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.40 HAL\_TIM\_PWM\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (</b> <b><i>TIM_HandleTypeDef</i> * 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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.41 HAL\_TIM\_PWM\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</b>
Function Description	Starts the TIM PWM signal generation in DMA mode.

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:           <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> TIM Channel 2 selected</li> <li>– <b>TIM_CHANNEL_3</b> TIM Channel 3 selected</li> <li>– <b>TIM_CHANNEL_4</b> TIM Channel 4 selected</li> </ul> </li> <li><b>pData</b> : : The source Buffer address.</li> <li><b>Length</b> : : The length of data to be transferred from memory to TIM peripheral</li> </ul>
Return values	<b>HAL status</b>
Notes	None.

### 38.2.42 HAL\_TIM\_PWM\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the TIM PWM signal generation in DMA 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:           <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> TIM Channel 2 selected</li> <li>– <b>TIM_CHANNEL_3</b> TIM Channel 3 selected</li> <li>– <b>TIM_CHANNEL_4</b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<b>HAL status</b>
Notes	None.

### 38.2.43 HAL\_TIM\_IC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Init (</b> <b><i>TIM_HandleTypeDef</i> *</b> <b>htim)</b>
Function Description	Initializes the TIM Input Capture Time base according to the specified parameters in the <i>TIM_HandleTypeDef</i> and create the associated handle.

Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : : TIM Input Capture handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.44 HAL\_TIM\_IC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_DeInit ( <i>TIM_HandleTypeDef</i> * <b>htim</b>)</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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.45 HAL\_TIM\_IC\_MspInit

Function Name	<b>void HAL_TIM_IC_MspInit ( <i>TIM_HandleTypeDef</i> * <b>htim</b>)</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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.46 HAL\_TIM\_IC\_MspDeInit

Function Name	<b>void HAL_TIM_IC_MspDeInit ( <i>TIM_HandleTypeDef</i> * <b>htim</b>)</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> |
| Notes         | <ul style="list-style-type: none"><li>• None.</li></ul> |

### 38.2.47 HAL\_TIM\_IC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, uint32_t <i>Channel</i>)</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:<ul style="list-style-type: none"><li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li><li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li><li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li><li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.48 HAL\_TIM\_IC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, uint32_t <i>Channel</i>)</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:<ul style="list-style-type: none"><li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li><li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li><li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li><li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.49 HAL\_TIM\_IC\_Start\_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIM_IC_Start_IT (     <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</code>
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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.50 HAL\_TIM\_IC\_Stop\_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (     <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</code>
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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.51 HAL\_TIM\_IC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel, uint32_t *</b> <b>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: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>– <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>– <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.52 HAL\_TIM\_IC\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops 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 disabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>– <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>– <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.53 HAL\_TIM\_OnePulse\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Init (</b>
---------------	--------------------------------------------------

***TIM\_HandleTypeDef* \* htim, uint32\_t OnePulseMode)**

Function Description	Initializes the TIM One Pulse Time Base according to the specified parameters in the <i>TIM_HandleTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM OnePulse handle</li> <li>• <b>OnePulseMode</b> : : Select the One pulse mode. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_OPmode_SINGLE</i></b> Only one pulse will be generated.</li> <li>- <b><i>TIM_OPmode_REPEATITIVE</i></b> Repetitive pulses wil be generated.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.54 HAL\_TIM\_OnePulse\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_DelInit (</b> <b><i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Delinitializes the TIM One Pulse.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM One Pulse handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.55 HAL\_TIM\_OnePulse\_MspInit

Function Name	<b>void HAL_TIM_OnePulse_MspInit (</b> <b><i>TIM_HandleTypeDef</i> *</b> <b>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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.56 HAL\_TIM\_OnePulse\_MspDeInit

Function Name	<code>void HAL_TIM_OnePulse_MspDeInit ( <i>TIM_HandleTypeDef</i> * htim)</code>
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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.57 HAL\_TIM\_OnePulse\_Start

Function Name	<code>HAL_StatusTypeDef HAL_TIM_OnePulse_Start (</code> <code><i>TIM_HandleTypeDef</i> * htim, uint32_t OutputChannel)</code>
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: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.58 HAL\_TIM\_OnePulse\_Stop

Function Name	<code>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (</code> <code><i>TIM_HandleTypeDef</i> * htim, uint32_t OutputChannel)</code>
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: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.59 HAL\_TIM\_OnePulse\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (</b> <b><i>TIM_HandleTypeDef</i> * 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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.60 HAL\_TIM\_OnePulse\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (</b> <b><i>TIM_HandleTypeDef</i> * 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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.61 HAL\_TIM\_Encoder\_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_Init (</code> <code>TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)</code>
Function Description	Initializes the TIM Encoder Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <code>htim</code> : : TIM Encoder Interface handle</li> <li>• <code>sConfig</code> : : TIM Encoder Interface configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.62 HAL\_TIM\_Encoder\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (</code> <code>TIM_HandleTypeDef * htim)</code>
Function Description	Deinitializes the TIM Encoder interface.
Parameters	<ul style="list-style-type: none"> <li>• <code>htim</code> : : TIM Encoder handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.63 HAL\_TIM\_Encoder\_MspInit

Function Name	<code>void HAL_TIM_Encoder_MspInit (</code> <code>TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> <li>• <code>htim</code> : : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.64 HAL\_TIM\_Encoder\_MspDeInit

Function Name	<code>void HAL_TIM_Encoder_MspDeInit ( <i>TIM_HandleTypeDef</i> * htim)</code>
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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.65 HAL\_TIM\_Encoder\_Start

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_Start (</code> <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
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: <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.66 HAL\_TIM\_Encoder\_Stop

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_Stop (</code> <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)
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: <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> </ul> </li> </ul>

	<ul style="list-style-type: none"><li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.67 HAL\_TIM\_Encoder\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (</b> <b><i>TIM_HandleTypeDef</i> * 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:<ul style="list-style-type: none"><li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li><li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.68 HAL\_TIM\_Encoder\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (</b> <b><i>TIM_HandleTypeDef</i> * 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:<ul style="list-style-type: none"><li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li><li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.69 HAL\_TIM\_Encoder\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (</code> <code>TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t *</code> <code>pData1, uint32_t * pData2, uint16_t Length)</code>
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: <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> TIM Channel 2 selected</li> </ul> </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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.70 HAL\_TIM\_Encoder\_Stop\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (</code> <code>TIM_HandleTypeDef * htim, uint32_t Channel)</code>
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: <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.71 HAL\_TIM\_IRQHandler

Function Name	<b>void HAL_TIM_IRQHandler ( <i>TIM_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.72 HAL\_TIM\_OC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel ( <i>TIM_HandleTypeDef</i> * htim, <i>TIM_OC_InitTypeDef</i> * sConfig, uint32_t Channel)</b>
Function Description	Initializes the TIM Output Compare Channels according to the specified parameters in the <i>TIM_OC_InitTypeDef</i> .
Parameters	<ul style="list-style-type: none"> <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: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>– <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>– <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.73 HAL\_TIM\_IC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel ( <i>TIM_HandleTypeDef</i> * htim, <i>TIM_IC_InitTypeDef</i> * sConfig, uint32_t Channel)</b>
Function Description	Initializes the TIM Input Capture Channels according to the specified parameters in the <i>TIM_IC_InitTypeDef</i> .
Parameters	<ul style="list-style-type: none"> <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:</li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.74 HAL\_TIM\_PWM\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>TIM_OC_InitTypeDef</i> * sConfig,</b> <b>uint32_t Channel)</b>
Function Description	Initializes the TIM PWM channels according to the specified parameters in the <i>TIM_OC_InitTypeDef</i> .
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle</li> <li>• <b>sConfig</b> : : TIM PWM configuration structure</li> <li>• <b>Channel</b> : : TIM Channels to be enabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.75 HAL\_TIM\_OnePulse\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>TIM_OnePulse_InitTypeDef</i> * sConfig, uint32_t OutputChannel, uint32_t InputChannel)</b>
Function Description	Initializes the TIM One Pulse Channels according to the specified parameters in the <i>TIM_OnePulse_InitTypeDef</i> .
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM One Pulse handle</li> <li>• <b>sConfig</b> : : TIM One Pulse configuration structure</li> <li>• <b>OutputChannel</b> : : TIM Channels to be enabled This parameter can be one of the following values:</li> </ul>

- ***TIM\_CHANNEL\_1*** TIM Channel 1 selected
  - ***TIM\_CHANNEL\_2*** TIM Channel 2 selected
  - **InputChannel** : : TIM Channels to be enabled This parameter can be one of the following values:
    - ***TIM\_CHANNEL\_1*** TIM Channel 1 selected
    - ***TIM\_CHANNEL\_2*** TIM Channel 2 selected
- Return values
- **HAL status**
- Notes
- None.

### 38.2.76 HAL\_TIM\_DMABurst\_WriteStart

Function Name	<code>HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (   <i>TIM_HandleTypeDef</i> * htim, uint32_t BurstBaseAddress,   uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t   BurstLength)</code>
Function Description	Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle</li> <li>• <b>BurstBaseAddress</b> : : TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <b><i>TIM_DMABase_CR1</i></b></li> <li>- <b><i>TIM_DMABase_CR2</i></b></li> <li>- <b><i>TIM_DMABase_SMCR</i></b></li> <li>- <b><i>TIM_DMABase_DIER</i></b></li> <li>- <b><i>TIM_DMABase_SR</i></b></li> <li>- <b><i>TIM_DMABase_EGR</i></b></li> <li>- <b><i>TIM_DMABase_CCMR1</i></b></li> <li>- <b><i>TIM_DMABase_CCMR2</i></b></li> <li>- <b><i>TIM_DMABase_CCER</i></b></li> <li>- <b><i>TIM_DMABase_CNT</i></b></li> <li>- <b><i>TIM_DMABase_PSC</i></b></li> <li>- <b><i>TIM_DMABase_ARR</i></b></li> <li>- <b><i>TIM_DMABase_RCR</i></b></li> <li>- <b><i>TIM_DMABase_CCR1</i></b></li> <li>- <b><i>TIM_DMABase_CCR2</i></b></li> <li>- <b><i>TIM_DMABase_CCR3</i></b></li> <li>- <b><i>TIM_DMABase_CCR4</i></b></li> <li>- <b><i>TIM_DMABase_BDTR</i></b></li> <li>- <b><i>TIM_DMABase_DCR</i></b></li> </ul> </li> <li>• <b>BurstRequestSrc</b> : : TIM DMA Request sources This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <b><i>TIM_DMA_UPDATE</i></b> TIM update Interrupt source</li> <li>- <b><i>TIM_DMA_CC1</i></b> TIM Capture Compare 1 DMA source</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b><i>TIM_DMA_CC2</i></b> TIM Capture Compare 2 DMA source</li> <li>- <b><i>TIM_DMA_CC3</i></b> TIM Capture Compare 3 DMA source</li> <li>- <b><i>TIM_DMA_CC4</i></b> TIM Capture Compare 4 DMA source</li> <li>- <b><i>TIM_DMA_COM</i></b> TIM Commutation DMA source</li> <li>- <b><i>TIM_DMA_TRIGGER</i></b> TIM Trigger DMA source</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.77 HAL\_TIM\_DMABurst\_WriteStop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t BurstRequestSrc)</b>
Function Description	Stops the TIM DMA Burst mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle</li> <li>• <b>BurstRequestSrc</b> : : TIM DMA Request sources to disable</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.78 HAL\_TIM\_DMABurst\_ReadStart

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t BurstBaseAddress,</b> <b>uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t</b> <b>BurstLength)</b>
Function Description	Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle</li> <li>• <b>BurstBaseAddress</b> : : TIM Base address from where the DMA will starts the Data read This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_DMABase_CR1</i></b></li> <li>- <b><i>TIM_DMABase_CR2</i></b></li> </ul> </li> </ul>

- *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 | • <b>HAL status</b> |
| Notes         | • None.             |

### 38.2.79 HAL\_TIM\_DMABurst\_ReadStop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (</b> <i>TIM_HandleTypeDef</i> * <i>htim</i> , <i>uint32_t</i> <i>BurstRequestSrc</i> )
Function Description	Stop the DMA burst reading.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle</li> <li>• <b>BurstRequestSrc</b> : : TIM DMA Request sources to disable.</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 38.2.80 HAL\_TIM\_GenerateEvent

Function Name	<b>HAL_StatusTypeDef HAL_TIM_GenerateEvent (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t EventSource)</b>
Function Description	Generate a software event.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle</li> <li>• <b>EventSource</b> : : specifies the event source. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_EventSource_Update</i></b> Timer update Event source</li> <li>– <b><i>TIM_EventSource_CC1</i></b> Timer Capture Compare 1 Event source</li> <li>– <b><i>TIM_EventSource_CC2</i></b> Timer Capture Compare 2 Event source</li> <li>– <b><i>TIM_EventSource_CC3</i></b> Timer Capture Compare 3 Event source</li> <li>– <b><i>TIM_EventSource_CC4</i></b> Timer Capture Compare 4 Event source</li> <li>– <b><i>TIM_EventSource_COM</i></b> Timer COM event source</li> <li>– <b><i>TIM_EventSource_Trigger</i></b> Timer Trigger Event source</li> <li>– <b><i>TIM_EventSource_Break</i></b> Timer Break event source</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• TIM6 and TIM7 can only generate an update event.</li> <li>• <b><i>TIM_EventSource_COM</i></b> and <b><i>TIM_EventSource_Break</i></b> are used only with TIM1, TIM15, TIM16 and TIM17.</li> </ul>

### 38.2.81 HAL\_TIM\_ConfigOCrefClear

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>TIM_ClearInputConfigTypeDef</i> * sClearInputConfig, uint32_t Channel)</b>
Function Description	Configures the OCRef clear feature.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle</li> <li>• <b>sClearInputConfig</b> : : pointer to a <b><i>TIM_ClearInputConfigTypeDef</i></b> structure that contains the OCREF clear feature and parameters for the TIM peripheral.</li> <li>• <b>Channel</b> : : specifies the TIM Channel This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_Channel_1</i></b> TIM Channel 1</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b><i>TIM_Channel_2</i></b> TIM Channel 2</li> <li>- <b><i>TIM_Channel_3</i></b> TIM Channel 3</li> <li>- <b><i>TIM_Channel_4</i></b> TIM Channel 4</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.82 HAL\_TIM\_ConfigClockSource

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigClockSource (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>TIM_ClockConfigTypeDef</i> * sClockSourceConfig)</b>
Function Description	Configures the clock source to be used.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle</li> <li>• <b>sClockSourceConfig</b> : : pointer to a <i>TIM_ClockConfigTypeDef</i> structure that contains the clock source information for the TIM peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.83 HAL\_TIM\_ConfigTI1Input

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>uint32_t</i> TI1_Selection)</b>
Function Description	Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle.</li> <li>• <b>TI1_Selection</b> : : Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_TI1SELECTION_CH1</i></b> The TIMx_CH1 pin is connected to TI1 input</li> <li>- <b><i>TIM_TI1SELECTION_XORCOMBINATION</i></b> The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)</li> </ul> </li> </ul>

---

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

### 38.2.84 HAL\_TIM\_SlaveConfigSynchronization

Function Name	<b>HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>TIM_SlaveConfigTypeDef</i> * sSlaveConfig)</b>
Function Description	Configures the TIM in Slave mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle.</li> <li>• <b>sSlaveConfig</b> : : pointer to a <i>TIM_SlaveConfigTypeDef</i> 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).</li> </ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.85 HAL\_TIM\_SlaveConfigSynchronization\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>TIM_SlaveConfigTypeDef</i> * sSlaveConfig)</b>
Function Description	Configures the TIM in Slave mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : TIM handle.</li> <li>• <b>sSlaveConfig</b> : pointer to a <i>TIM_SlaveConfigTypeDef</i> 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).</li> </ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.86 HAL\_TIM\_ReadCapturedValue

Function Name	<code>uint32_t HAL_TIM_ReadCapturedValue ( <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</code>
Function Description	Read the captured value from Capture Compare unit.
Parameters	<ul style="list-style-type: none"> <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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Captured value</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.87 HAL\_TIM\_PeriodElapsedCallback

Function Name	<code>void HAL_TIM_PeriodElapsedCallback ( <i>TIM_HandleTypeDef</i> * htim)</code>
Function Description	Period elapsed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.88 HAL\_TIM\_OC\_DelayElapsedCallback

Function Name	<code>void HAL_TIM_OC_DelayElapsedCallback ( <i>TIM_HandleTypeDef</i> * htim)</code>
Function Description	Output Compare callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM OC handle</li> </ul>

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

### 38.2.89 HAL\_TIM\_IC\_CaptureCallback

Function Name	<b>void HAL_TIM_IC_CaptureCallback ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Input Capture callback in non blocking mode.
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>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.90 HAL\_TIM\_PWM\_PulseFinishedCallback

Function Name	<b>void HAL_TIM_PWM_PulseFinishedCallback ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	PWM Pulse finished 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.91 HAL\_TIM\_TriggerCallback

Function Name	<b>void HAL_TIM_TriggerCallback ( <i>TIM_HandleTypeDef</i> * htim)</b>
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>

## Notes

- None.

### 38.2.92 HAL\_TIM\_ErrorCallback

Function Name	<b>void HAL_TIM_ErrorCallback ( <i>TIM_HandleTypeDef</i> * 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.93 HAL\_TIM\_Base\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_Base_GetState ( <i>TIM_HandleTypeDef</i> * 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>• <b>HAL state</b></li></ul>

### 38.2.94 HAL\_TIM\_OC\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_OC_GetState ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Return the TIM OC state.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : : TIM Ouput Compare handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>

### 38.2.95 HAL\_TIM\_PWM\_GetState

Function Name	<code>HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (     <i>TIM_HandleTypeDef</i> * htim)</code>
Function Description	Return the TIM PWM state.
Parameters	<ul style="list-style-type: none"><li>• <code>htim</code> : : TIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.96 HAL\_TIM\_IC\_GetState

Function Name	<code>HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (     <i>TIM_HandleTypeDef</i> * htim)</code>
Function Description	Return the TIM Input Capture state.
Parameters	<ul style="list-style-type: none"><li>• <code>htim</code> : : TIM IC handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.97 HAL\_TIM\_OnePulse\_GetState

Function Name	<code>HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (     <i>TIM_HandleTypeDef</i> * htim)</code>
Function Description	Return the TIM One Pulse Mode state.
Parameters	<ul style="list-style-type: none"><li>• <code>htim</code> : : TIM OPM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 38.2.98 HAL\_TIM\_Encoder\_GetState

Function Name	<code>HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState ( TIM_HandleTypeDef * htim)</code>
Function Description	Return the TIM Encoder Mode state.
Parameters	<ul style="list-style-type: none"><li>• <code>htim</code> : : TIM Encoder handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 38.3 TIM Firmware driver defines

### 38.3.1 TIM

TIM

***TIM Automatic Output Enable***

`TIM_AUTOMATICOUTPUT_ENABLE`

`TIM_AUTOMATICOUTPUT_DISABLE`

`IS_TIM_AUTOMATIC_OUTPUT_STATE`

***TIM Break Input Enable***

`TIM_BREAK_ENABLE`

`TIM_BREAK_DISABLE`

`IS_TIM_BREAK_STATE`

***TIM Break Input Polarity***

`TIM_BREAKPOLARITY_LOW`

`TIM_BREAKPOLARITY_HIGH`

`IS_TIM_BREAK_POLARITY`

***TIM Channel***

`TIM_CHANNEL_1`

`TIM_CHANNEL_2`

`TIM_CHANNEL_3`

`TIM_CHANNEL_4`

TIM\_CHANNEL\_ALL  
IS\_TIM\_CHANNELS  
IS\_TIM\_PWMI\_CHANNELS  
IS\_TIM\_OPM\_CHANNELS  
IS\_TIM\_COMPLEMENTARY\_CHANNELS

***TIM Clear Input Filter***

IS\_TIM\_CLEARINPUT\_FILTER

***TIM Clear Input Polarity***

TIM\_CLEARINPUTPOLARITY\_INVERTED      Polarity for ETRx pin

TIM\_CLEARINPUTPOLARITY\_NONINVERTED      Polarity for ETRx pin

IS\_TIM\_CLEARINPUT\_POLARITY

***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.

IS\_TIM\_CLEARINPUT\_PRESCALER

***TIM ClearInput Source***

TIM\_CLEARINPUTSOURCE\_ETR

TIM\_CLEARINPUTSOURCE\_NONE

IS\_TIM\_CLEARINPUT\_SOURCE

***TIM Clock Division***

TIM\_CLOCKDIVISION\_DIV1

TIM\_CLOCKDIVISION\_DIV2

TIM\_CLOCKDIVISION\_DIV4

IS\_TIM\_CLOCKDIVISION\_DIV

***TIM Clock Filter***

IS\_TIM\_CLOCKFILTER

IS\_TIM\_DEADTIME

***TIM Clock Polarity***

TIM\_CLOCKPOLARITY\_INVERTED      Polarity for ETRx clock sources

TIM\_CLOCKPOLARITY\_NONINVERTED      Polarity for ETRx clock sources

TIM\_CLOCKPOLARITY\_RISING      Polarity for TIx clock sources

TIM\_CLOCKPOLARITY\_FALLING      Polarity for TIx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE	Polarity for TIx clock sources
IS_TIM_CLOCKPOLARITY	
<b>TIM Clock Prescaler</b>	
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.
IS_TIM_CLOCKPRESCALER	
<b>TIM Clock Source</b>	
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	
IS_TIM_CLOCKSOURCE	
<b>TIM Commutation</b>	
TIM_COMMUTATION_TRGI	
TIM_COMMUTATION_SOFTWARE	
<b>TIM Counter Mode</b>	
TIM_COUNTERMODE_UP	
TIM_COUNTERMODE_DOWN	
TIM_COUNTERMODE_CENTERALIGNED1	
TIM_COUNTERMODE_CENTERALIGNED2	
TIM_COUNTERMODE_CENTERALIGNED3	
IS_TIM_COUNTER_MODE	
<b>TIM DMA Base address</b>	
TIM_DMABase_CR1	
TIM_DMABase_CR2	
TIM_DMABase_SMCR	

TIM\_DMABase\_DIER  
TIM\_DMABase\_SR  
TIM\_DMABase\_EGR  
TIM\_DMABase\_CCMR1  
TIM\_DMABase\_CCMR2  
TIM\_DMABase\_CCER  
TIM\_DMABase\_CNT  
TIM\_DMABase\_PSC  
TIM\_DMABase\_ARR  
TIM\_DMABase\_RCR  
TIM\_DMABase\_CCR1  
TIM\_DMABase\_CCR2  
TIM\_DMABase\_CCR3  
TIM\_DMABase\_CCR4  
TIM\_DMABase\_BDTR  
TIM\_DMABase\_DCR  
TIM\_DMABase\_OR  
IS\_TIM\_DMA\_BASE

***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**

**IS\_TIM\_DMA\_LENGTH**

***TIM DMA Handle Index***

**TIM\_DMA\_ID\_UPDATE** Index of the DMA handle used for Update DMA requests

**TIM\_DMA\_ID\_CC1** Index of the DMA handle used for Capture/Compare 1 DMA requests

**TIM\_DMA\_ID\_CC2** Index of the DMA handle used for Capture/Compare 2 DMA requests

**TIM\_DMA\_ID\_CC3** Index of the DMA handle used for Capture/Compare 3 DMA requests

**TIM\_DMA\_ID\_CC4** Index of the DMA handle used for Capture/Compare 4 DMA requests

**TIM\_DMA\_ID\_COMMUTATION** Index of the DMA handle used for Commutation DMA requests

**TIM\_DMA\_ID\_TRIGGER** Index of the DMA handle used for Trigger DMA requests

***TIM DMA Sources***

**TIM\_DMA\_UPDATE**

**TIM\_DMA\_CC1**

**TIM\_DMA\_CC2**

**TIM\_DMA\_CC3**

**TIM\_DMA\_CC4**

**TIM\_DMA\_COM**

**TIM\_DMA\_TRIGGER**

**IS\_TIM\_DMA\_SOURCE**

***TIM Encoder Mode***

**TIM\_ENCODERMODE\_TI1**

**TIM\_ENCODERMODE\_TI2**

**TIM\_ENCODERMODE\_TI12**

**IS\_TIM\_ENCODER\_MODE**

***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  
IS\_TIM\_EVENT\_SOURCE

***TIM Exported Macros***

<code>__HAL_TIM_RESET_HANDLE_STATE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Reset TIM handle state.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: TIM handle.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None:</li></ul>
<code>__HAL_TIM_ENABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Enable the TIM peripheral.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: TIM handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None:</li></ul>
<code>__HAL_TIM_MOE_ENABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Enable the TIM main Output.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: TIM handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None:</li></ul>
<code>__HAL_TIM_DISABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Disable the TIM peripheral.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: TIM handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None:</li></ul>
<code>__HAL_TIM_MOE_DISABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Disable the TIM main Output.</li></ul>

**Parameters:**

- `__HANDLE__`: TIM handle

**Return value:**

- None:

`__HAL_TIM_ENABLE_IT`  
`__HAL_TIM_ENABLE_DMA`  
`__HAL_TIM_DISABLE_IT`  
`__HAL_TIM_DISABLE_DMA`  
`__HAL_TIM_GET_FLAG`  
`__HAL_TIM_CLEAR_FLAG`  
`__HAL_TIM_GET_ITSTATUS`  
`__HAL_TIM_CLEAR_IT`  
`__HAL_TIM_DIRECTION_STATUS`  
`__HAL_TIM_PRESCALER`  
`__HAL_TIM_SetICPrescalerValue`  
`__HAL_TIM_ResetICPrescalerValue`  
`__HAL_TIM_SetCompare`

**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_GetCompare`

**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:

`__HAL_TIM_SetCounter`

**Description:**

- Sets the TIM Counter Register value on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__COUNTER__`: specifies the Counter register new value.

**Return value:**

- None:

`__HAL_TIM_GetCounter`

**Description:**

- Gets the TIM Counter Register value on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.

**Return value:**

- None:

`__HAL_TIM_SetAutoreload`

**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:

`__HAL_TIM_GetAutoreload`

**Description:**

- Gets the TIM Autoreload Register value on

runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.

**Return value:**

- None:

`__HAL_TIM_SetClockDivision`

**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_GetClockDivision`

**Description:**

- Gets the TIM Clock Division value on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.

**Return value:**

- None:

`__HAL_TIM_SetICPrescaler`

**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_GetICPrescaler`

**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:

***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

IS\_TIM\_FLAG

***TIM Input Capture Value***

IS\_TIM\_IC\_FILTER

***TIM Input Capture Polarity***

TIM\_ICPOLARITY\_RISING

TIM\_ICPOLARITY\_FALLING

TIM\_ICPOLARITY\_BOTHEDGE

IS\_TIM\_IC\_POLARITY

***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

IS\_TIM\_IC\_PRESCALER

***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

IS\_TIM\_IC\_SELECTION

***TIM Input Channel polarity***

TIM\_INPUTCHANNELPOLARITY\_RISING      Polarity for TIx source

TIM\_INPUTCHANNELPOLARITY\_FALLING      Polarity for TIx source

TIM\_INPUTCHANNELPOLARITY\_BOTHEDGE Polarity for TIx source

***TIM interrupt Definition***

TIM\_IT\_UPDATE

TIM\_IT\_CC1

TIM\_IT\_CC2

TIM\_IT\_CC3

TIM\_IT\_CC4

TIM\_IT\_COM

TIM\_IT\_TRIGGER

TIM\_IT\_BREAK

***TIM Lock Configuration***

TIM\_LOCKLEVEL\_OFF

TIM\_LOCKLEVEL\_1

TIM\_LOCKLEVEL\_2

TIM\_LOCKLEVEL\_3

IS\_TIM\_LOCK\_LEVEL

***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

IS\_TIM\_TRGO\_SOURCE

***TIM Master/Slave Mode***

TIM\_MASTERSLAVEMODE\_ENABLE

TIM\_MASTERSLAVEMODE\_DISABLE

IS\_TIM\_MSM\_STATE

***TIM One Pulse Mode***

TIM\_OPMODE\_SINGLE

TIM\_OPMODE\_REPETITIVE

IS\_TIM\_OPM\_MODE

***TIM Off-state Selection for Idle Mode***

TIM\_OSSI\_ENABLE

TIM\_OSSI\_DISABLE

IS\_TIM\_OSSI\_STATE

***TIM Off-state Selection for Run Mode***

TIM\_OSSR\_ENABLE

TIM\_OSSR\_DISABLE

IS\_TIM\_OSSR\_STATE

***TIM Output Compare & 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

IS\_TIM\_PWM\_MODE

IS\_TIM\_OC\_MODE

***TIM Output Compare Idle State***

TIM\_OCIDLESTATE\_SET

TIM\_OCIDLESTATE\_RESET

IS\_TIM\_OCIDLE\_STATE

***TIM Complementary Output Compare Idle State***

TIM\_OCNIDLESTATE\_SET

TIM\_OCNIDLESTATE\_RESET

IS\_TIM\_OCNIDLE\_STATE

***TIM Complementary Output Compare Polarity***

TIM\_OCNPOLARITY\_HIGH

TIM\_OCNPOLARITY\_LOW

IS\_TIM\_OCN\_POLARITY

***TIM Complementary Output Compare State***

TIM\_OUTPUTNSTATE\_DISABLE

TIM\_OUTPUTNSTATE\_ENABLE

IS\_TIM\_OUTPUTN\_STATE

***TIM Output Compare Polarity***

TIM\_OCPOLARITY\_HIGH

TIM\_OCPOLARITY\_LOW

IS\_TIM\_OC\_POLARITY  
**TIM Output Compare State**  
TIM\_OUTPUTSTATE\_DISABLE  
TIM\_OUTPUTSTATE\_ENABLE  
IS\_TIM\_OUTPUT\_STATE  
**TIM Output Fast State**  
TIM\_OCFAST\_DISABLE  
TIM\_OCFAST\_ENABLE  
IS\_TIM\_FAST\_STATE  
**TIM Private Macros**  
CCER\_CCxE\_MASK  
CCER\_CCxNE\_MASK  
**TIM Slave Mode**  
TIM\_SLAVERESET\_DISABLE  
TIM\_SLAVERESET\_RESET  
TIM\_SLAVERESET\_GATED  
TIM\_SLAVERESET\_TRIGGER  
TIM\_SLAVERESET\_EXTERNAL1  
IS\_TIM\_SLAVE\_MODE  
**TIM TI1 Input Selection**  
TIM\_TI1SELECTION\_CH1  
TIM\_TI1SELECTION\_XORCOMBINATION  
IS\_TIM\_TI1SELECTION  
**TIM Trigger Filter**  
IS\_TIM\_TRIGGERFILTER  
**TIM Trigger Polarity**  
TIM\_TRIGGERPOLARITY\_INVERTED      Polarity for ETRx trigger sources  
TIM\_TRIGGERPOLARITY\_NONINVERTED      Polarity for ETRx trigger sources  
TIM\_TRIGGERPOLARITY\_RISING      Polarity for TIxFPx or TI1\_ED trigger sources  
TIM\_TRIGGERPOLARITY\_FALLING      Polarity for TIxFPx or TI1\_ED trigger sources  
TIM\_TRIGGERPOLARITY\_BOTHEDGE      Polarity for TIxFPx or TI1\_ED trigger sources  
IS\_TIM\_TRIGGERPOLARITY  
**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.

**IS\_TIM\_TRIGGERPRESCALER*****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

IS\_TIM\_TRIGGER\_SELECTION

IS\_TIM\_INTERNAL\_TRIGGER\_SELECTION

IS\_TIM\_INTERNAL\_TRIGGEREVENT\_SELECTION

## 39 HAL TIM Extension Driver

### 39.1 TIMEx Firmware driver registers structures

#### 39.1.1 TIM\_HallSensor\_InitTypeDef

*TIM\_HallSensor\_InitTypeDef* is defined in the `stm32f0xx_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

#### 39.1.2 TIM\_MasterConfigTypeDef

*TIM\_MasterConfigTypeDef* is defined in the `stm32f0xx_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](#)

#### 39.1.3 TIM\_BreakDeadTimeConfigTypeDef

*TIM\_BreakDeadTimeConfigTypeDef* is defined in the `stm32f0xx_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\*](#)

## 39.2 TIMEx Firmware driver API description

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

### 39.2.1 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

### 39.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
  - Complementary Output Compare : `HAL_TIM_OC_MspInit()`
  - Complementary PWM generation : `HAL_TIM_PWM_MspInit()`

- Complementary One-pulse mode output : HAL\_TIM\_OnePulse\_MspInit()
  - Hall Sensor output : HAL\_TIM\_HallSensor\_MspInit()
2. Initialize the TIM low level resources :
    - a. Enable the TIM interface clock using \_\_TIMx\_CLK\_ENABLE();
    - b. TIM pins configuration
      - Enable the clock for the TIM GPIOs using the following function:  
\_\_GPIOx\_CLK\_ENABLE();
      - Configure these TIM pins in Alternate function mode using  
HAL\_GPIO\_Init();
  3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL\_TIM\_ConfigClockSource, the clock configuration should be done before any start function.
  4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
    - HAL\_TIMEx\_HallSensor\_Init and HAL\_TIMEx\_ConfigCommutationEvent: to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).
  5. Activate the TIM peripheral using one of the start functions:
    - Complementary Output Compare : HAL\_TIMEx\_OCN\_Start(),  
HAL\_TIMEx\_OCN\_Start\_DMA(), HAL\_TIMEx\_OCN\_Start\_IT()
    - Complementary PWM generation : HAL\_TIMEx\_PWMN\_Start(),  
HAL\_TIMEx\_PWMN\_Start\_DMA(), HAL\_TIMEx\_PWMN\_Start\_IT()
    - Complementary One-pulse mode output : HAL\_TIMEx\_OnePulseN\_Start(),  
HAL\_TIMEx\_OnePulseN\_Start\_IT()
    - Hall Sensor output : HAL\_TIMEx\_HallSensor\_Start(),  
HAL\_TIMEx\_HallSensor\_Start\_DMA(), HAL\_TIMEx\_HallSensor\_Start\_IT().

### 39.2.3 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\(\)\*\*](#)

### 39.2.4 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\(\)`](#)

### 39.2.5 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\(\)`](#)

### 39.2.6 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()`

### 39.2.7 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.
- Configure timer remapping capabilities.
- `HAL_TIMEx_ConfigCommutationEvent()`
- `HAL_TIMEx_ConfigCommutationEvent_IT()`
- `HAL_TIMEx_ConfigCommutationEvent_DMA()`
- `HAL_TIMEx_MasterConfigSynchronization()`
- `HAL_TIMEx_ConfigBreakDeadTime()`
- `HAL_TIMEx_RemapConfig()`

### 39.2.8 HAL\_TIMEx\_HallSensor\_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (</code> <code>TIM_HandleTypeDef * htim, TIM_HallSensor_InitTypeDef * sConfig)</code>
Function Description	Initializes the TIM Hall Sensor Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <code>htim</code> : : TIM Encoder Interface handle</li> <li>• <code>sConfig</code> : : TIM Hall Sensor configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.9 HAL\_TIMEx\_HallSensor\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_HallSensor_DelInit (</code> <code>TIM_HandleTypeDef * htim)</code>
Function Description	Delinitializes the TIM Hall Sensor interface.
Parameters	<ul style="list-style-type: none"> <li>• <code>htim</code> : : TIM Hall Sensor handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

## Notes

- None.

### 39.2.10 HAL\_TIMEx\_HallSensor\_MspInit

Function Name	<b>void HAL_TIMEx_HallSensor_MspInit ( <i>TIM_HandleTypeDef</i> * <i>htim</i>)</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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 39.2.11 HAL\_TIMEx\_HallSensor\_MspDeInit

Function Name	<b>void HAL_TIMEx_HallSensor_MspDeInit ( <i>TIM_HandleTypeDef</i> * <i>htim</i>)</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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 39.2.12 HAL\_TIMEx\_HallSensor\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start ( <i>TIM_HandleTypeDef</i> * <i>htim</i>)</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>• <b>HAL status</b></li></ul>

## Notes

- None.

### 39.2.13 HAL\_TIMEx\_HallSensor\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (</b> <b><i>TIM_HandleTypeDef * htim</i></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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 39.2.14 HAL\_TIMEx\_HallSensor\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 39.2.15 HAL\_TIMEx\_HallSensor\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
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>• <b>HAL status</b></li></ul>

## Notes

- None.

### 39.2.16 HAL\_TIMEx\_HallSensor\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t * pData, uint16_t Length)</b>
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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 39.2.17 HAL\_TIMEx\_HallSensor\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim)</b>
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>• <b>HAL status</b></li></ul>

### 39.2.18 HAL\_TIMEx\_OCN\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Start (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
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: <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> TIM Channel 2 selected</li> <li>– <b>TIM_CHANNEL_3</b> TIM Channel 3 selected</li> <li>– <b>TIM_CHANNEL_4</b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.19 HAL\_TIMEx\_OCN\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (</b> <b><i>TIM_HandleTypeDef</i> * 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: <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> TIM Channel 2 selected</li> <li>– <b>TIM_CHANNEL_3</b> TIM Channel 3 selected</li> <li>– <b>TIM_CHANNEL_4</b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.20 HAL\_TIMEx\_OCN\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (</b> <b><i>TIM_HandleTypeDef</i> * 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</li> </ul>

---

	be one of the following values:
	– <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected
	– <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected
	– <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected
	– <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected
Return values	• <b>HAL status</b>
Notes	• None.

### 39.2.21 HAL\_TIMEx\_OCN\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (</b> <b><i>TIM_HandleTypeDef</i> * 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 be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>– <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>– <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 39.2.22 HAL\_TIMEx\_OCN\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</b>
Function Description	Starts the TIM Output Compare signal generation in DMA 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 enabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul>
• <b>pData</b> :	The source Buffer address.
• <b>Length</b> :	The length of data to be transferred from memory to TIM peripheral
Return values	• <b>HAL status</b>
Notes	• None.

### 39.2.23 HAL\_TIMEx\_OCN\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Output Compare signal generation in DMA 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 be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 39.2.24 HAL\_TIMEx\_PWMN\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Starts the PWM 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 enabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.25 HAL\_TIMEx\_PWMN\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the PWM 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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.26 HAL\_TIMEx\_PWMN\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Starts the PWM signal generation in interrupt mode 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: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.27 HAL\_TIMEx\_PWMN\_Stop\_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (</code> <code>  <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</code>
Function Description	Stops the PWM signal generation in interrupt mode 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: <ul style="list-style-type: none"> <li>- <code>TIM_CHANNEL_1</code> TIM Channel 1 selected</li> <li>- <code>TIM_CHANNEL_2</code> TIM Channel 2 selected</li> <li>- <code>TIM_CHANNEL_3</code> TIM Channel 3 selected</li> <li>- <code>TIM_CHANNEL_4</code> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.28 HAL\_TIMEx\_PWMN\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (</code> <code>  <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel, uint32_t *</code> <code>  pData, uint16_t Length)</code>
Function Description	Starts the TIM PWM signal generation in DMA mode 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 enabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>TIM_CHANNEL_1</code> TIM Channel 1 selected</li> <li>- <code>TIM_CHANNEL_2</code> TIM Channel 2 selected</li> <li>- <code>TIM_CHANNEL_3</code> TIM Channel 3 selected</li> <li>- <code>TIM_CHANNEL_4</code> TIM Channel 4 selected</li> </ul> </li> <li>• <b>pData</b> : : The source Buffer address.</li> <li>• <b>Length</b> : : The length of data to be transferred from memory to TIM peripheral</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.29 HAL\_TIMEx\_PWMN\_Stop\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (     <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</code>
Function Description	Stops the TIM PWM signal generation in DMA mode 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: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> <li>– <b><i>TIM_CHANNEL_3</i></b> TIM Channel 3 selected</li> <li>– <b><i>TIM_CHANNEL_4</i></b> TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.30 HAL\_TIMEx\_OnePulseN\_Start

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (     <i>TIM_HandleTypeDef</i> * htim, uint32_t OutputChannel)</code>
Function Description	Starts the TIM One Pulse signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM One Pulse handle</li> <li>• <b>OutputChannel</b> : : TIM Channel to be enabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.31 HAL\_TIMEx\_OnePulseN\_Stop

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (</code> <code>TIM_HandleTypeDef * htim, uint32_t OutputChannel)</code>
Function Description	Stops the TIM One Pulse signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM One Pulse handle</li> <li>• <b>OutputChannel</b> : : TIM Channel to be disabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>TIM_CHANNEL_1</code> TIM Channel 1 selected</li> <li>- <code>TIM_CHANNEL_2</code> TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.32 HAL\_TIMEx\_OnePulseN\_Start\_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (</code> <code>TIM_HandleTypeDef * htim, uint32_t OutputChannel)</code>
Function Description	Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM One Pulse handle</li> <li>• <b>OutputChannel</b> : : TIM Channel to be enabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>TIM_CHANNEL_1</code> TIM Channel 1 selected</li> <li>- <code>TIM_CHANNEL_2</code> TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.33 HAL\_TIMEx\_OnePulseN\_Stop\_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (</code> <code>TIM_HandleTypeDef * htim, uint32_t OutputChannel)</code>
---------------	--------------------------------------------------------------------------------------------------------------------------------

Function Description	Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM One Pulse handle</li> <li>• <b>OutputChannel</b> : : TIM Channel to be disabled This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.34 HAL\_TIMEx\_ConfigCommutationEvent

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t InputTrigger, uint32_t</b> <b>CommutationSource)</b>
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: <ul style="list-style-type: none"> <li>– <b>TIM_TS_ITR0</b> Internal trigger 0 selected</li> <li>– <b>TIM_TS_ITR1</b> Internal trigger 1 selected</li> <li>– <b>TIM_TS_ITR2</b> Internal trigger 2 selected</li> <li>– <b>TIM_TS_ITR3</b> Internal trigger 3 selected</li> <li>– <b>TIM_TS_NONE</b> No trigger is needed</li> </ul> </li> <li>• <b>CommutationSource</b> : : the Commutation Event source This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>TIM_COMMUTATION_TRGI</b> Commutation source is the TRGI of the Interface Timer</li> <li>– <b>TIM_COMMUTATION_SOFTWARE</b> Commutation source is set by software using the COMG bit</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></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>

### 39.2.35 HAL\_TIMEx\_ConfigCommutationEvent\_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_IT (   <b>TIM_HandleTypeDef</b> * htim, uint32_t InputTrigger, uint32_t   CommutationSource)</code>
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: <ul style="list-style-type: none"> <li>- <b>TIM_TS_ITR0</b> Internal trigger 0 selected</li> <li>- <b>TIM_TS_ITR1</b> Internal trigger 1 selected</li> <li>- <b>TIM_TS_ITR2</b> Internal trigger 2 selected</li> <li>- <b>TIM_TS_ITR3</b> Internal trigger 3 selected</li> <li>- <b>TIM_TS_NONE</b> No trigger is needed</li> </ul> </li> <li>• <b>CommutationSource</b> : : the Commutation Event source This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>TIM_COMMUTATION_TRGI</b> Commutation source is the TRGI of the Interface Timer</li> <li>- <b>TIM_COMMUTATION_SOFTWARE</b> Commutation source is set by software using the COMG bit</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></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>

### 39.2.36 HAL\_TIMEx\_ConfigCommutationEvent\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_DMA (   <b>TIM_HandleTypeDef</b> * htim, uint32_t InputTrigger, uint32_t   CommutationSource)</code>
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: <ul style="list-style-type: none"> <li>- <b>TIM_TS_ITR0</b> Internal trigger 0 selected</li> <li>- <b>TIM_TS_ITR1</b> Internal trigger 1 selected</li> <li>- <b>TIM_TS_ITR2</b> Internal trigger 2 selected</li> <li>- <b>TIM_TS_ITR3</b> Internal trigger 3 selected</li> <li>- <b>TIM_TS_NONE</b> No trigger is needed</li> </ul> </li> <li><b>CommutationSource</b> : : the Commutation Event source This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>TIM_COMMUTATION_TRGI</b> Commutation source is the TRGI of the Interface Timer</li> <li>- <b>TIM_COMMUTATION_SOFTWARE</b> Commutation source is set by software using the COMG bit</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></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>

### 39.2.37 HAL\_TIMEx\_MasterConfigSynchronization

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization ( TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)</code>
Function Description	Configures the TIM in master mode.
Parameters	<ul style="list-style-type: none"> <li><b>htim</b> : : TIM handle.</li> <li><b>sMasterConfig</b> : : pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 39.2.38 HAL\_TIMEx\_ConfigBreakDeadTime

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (</code> <code>  <b>TIM_HandleTypeDef</b> * htim,</code> <code>  <b>TIM_BreakDeadTimeConfigTypeDef</b> *</code> <code>  <b>sBreakDeadTimeConfig</b>)</code>
Function Description	Configures the Break feature, dead time, Lock level, OSS1/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 <code>TIM_ConfigBreakDeadConfigTypeDef</code> structure that contains the BDTR Register configuration information for the TIM peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.39 HAL\_TIMEx\_RemapConfig

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_RemapConfig (</code> <code>  <b>TIM_HandleTypeDef</b> * htim, uint32_t Remap)</code>
Function Description	Configures the TIM14 Remapping input capabilities.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : : TIM handle.</li> <li>• <b>Remap</b> : : specifies the TIM remapping source. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>TIM_TIM14_GPIO</b> TIM14 TI1 is connected to GPIO</li> <li>- <b>TIM_TIM14_RTC</b> TIM14 TI1 is connected to RTC_clock</li> <li>- <b>TIM_TIM14_HSE</b> TIM14 TI1 is connected to HSE/32</li> <li>- <b>TIM_TIM14_MCO</b> TIM14 TI1 is connected to MCO</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.40 HAL\_TIMEx\_CommulationCallback

Function Name	<b>void HAL_TIMEx_CommutationCallback ( <i>TIM_HandleTypeDef</i> * <i>htim</i>)</b>
Function Description	Hall commutation changed 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 39.2.41 HAL\_TIMEx\_BreakCallback

Function Name	<b>void HAL_TIMEx_BreakCallback ( <i>TIM_HandleTypeDef</i> * <i>htim</i>)</b>
Function Description	Hall Break 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>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 39.2.42 HAL\_TIMEx\_DMACommutationCplt

Function Name	<b>void HAL_TIMEx_DMACommutationCplt ( <i>DMA_HandleTypeDef</i> * <i>hdma</i>)</b>
Function Description	TIM DMA Commutation callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma</b> : : pointer to DMA handle.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 39.2.43 HAL\_TIMEx\_HallSensor\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (</b>
---------------	-------------------------------------------------------------

*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>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 39.3 TIMEx Firmware driver defines

### 39.3.1 TIMEx

TIMEx

*TIMEx Remap*

TIM_TIM14_GPIO	TIM14 TI1 is connected to GPIO
TIM_TIM14_RTC	TIM14 TI1 is connected to RTC_clock
TIM_TIM14_HSE	TIM14 TI1 is connected to HSE/32
TIM_TIM14_MCO	TIM14 TI1 is connected to MCO
IS_TIM_REMAP	

## 40 HAL TSC Generic Driver

### 40.1 TSC Firmware driver registers structures

#### 40.1.1 TSC\_InitTypeDef

*TSC\_InitTypeDef* is defined in the `stm32f0xx_hal_tsc.h`

##### Data Fields

- *uint32\_t CTPulseHighLength*
- *uint32\_t CTPulseLowLength*
- *uint32\_t SpreadSpectrum*
- *uint32\_t SpreadSpectrumDeviation*
- *uint32\_t SpreadSpectrumPrescaler*
- *uint32\_t PulseGeneratorPrescaler*
- *uint32\_t MaxCountValue*
- *uint32\_t IODefaultMode*
- *uint32\_t SynchroPinPolarity*
- *uint32\_t AcquisitionMode*
- *uint32\_t MaxCountInterrupt*
- *uint32\_t ChannelIOs*
- *uint32\_t ShieldIOs*
- *uint32\_t SamplingIOs*

##### Field Documentation

- *uint32\_t TSC\_InitTypeDef::CTPulseHighLength* Charge-transfer high pulse length
- *uint32\_t TSC\_InitTypeDef::CTPulseLowLength* Charge-transfer low pulse length
- *uint32\_t TSC\_InitTypeDef::SpreadSpectrum* Spread spectrum activation
- *uint32\_t TSC\_InitTypeDef::SpreadSpectrumDeviation* Spread spectrum deviation
- *uint32\_t TSC\_InitTypeDef::SpreadSpectrumPrescaler* Spread spectrum prescaler
- *uint32\_t TSC\_InitTypeDef::PulseGeneratorPrescaler* Pulse generator prescaler
- *uint32\_t TSC\_InitTypeDef::MaxCountValue* Max count value
- *uint32\_t TSC\_InitTypeDef::IODefaultMode* IO default mode
- *uint32\_t TSC\_InitTypeDef::SynchroPinPolarity* Synchro pin polarity
- *uint32\_t TSC\_InitTypeDef::AcquisitionMode* Acquisition mode
- *uint32\_t TSC\_InitTypeDef::MaxCountInterrupt* Max count interrupt activation
- *uint32\_t TSC\_InitTypeDef::ChannelIOs* Channel IOs mask
- *uint32\_t TSC\_InitTypeDef::ShieldIOs* Shield IOs mask
- *uint32\_t TSC\_InitTypeDef::SamplingIOs* Sampling IOs mask

#### 40.1.2 TSC\_IOConfigTypeDef

*TSC\_IOConfigTypeDef* is defined in the `stm32f0xx_hal_tsc.h`

##### Data Fields

- *uint32\_t ChannelIOs*
- *uint32\_t ShieldIOs*
- *uint32\_t SamplingIOs*

#### Field Documentation

- *uint32\_t TSC\_IOConfigTypeDef::ChannelIOs* Channel IOs mask
- *uint32\_t TSC\_IOConfigTypeDef::ShieldIOs* Shield IOs mask
- *uint32\_t TSC\_IOConfigTypeDef::SamplingIOs* Sampling IOs mask

### 40.1.3 TSC\_HandleTypeDef

*TSC\_HandleTypeDef* is defined in the `stm32f0xx_hal_tsc.h`

#### Data Fields

- *TSC\_TypeDef \* Instance*
- *TSC\_InitTypeDef Init*
- *\_\_IO HAL\_TSC\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*

#### Field Documentation

- *TSC\_TypeDef\* TSC\_HandleTypeDef::Instance* Register base address
- *TSC\_InitTypeDef TSC\_HandleTypeDef::Init* Initialization parameters
- *\_\_IO HAL\_TSC\_StateTypeDef TSC\_HandleTypeDef::State* Peripheral state
- *HAL\_LockTypeDef TSC\_HandleTypeDef::Lock* Lock feature

## 40.2 TSC Firmware driver API description

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

### 40.2.1 TSC specific features

1. Proven and robust surface charge transfer acquisition principle
2. Supports up to 3 capacitive sensing channels per group
3. Capacitive sensing channels can be acquired in parallel offering a very good response time
4. Spread spectrum feature to improve system robustness in noisy environments
5. Full hardware management of the charge transfer acquisition sequence
6. Programmable charge transfer frequency
7. Programmable sampling capacitor I/O pin
8. Programmable channel I/O pin
9. Programmable max count value to avoid long acquisition when a channel is faulty
10. Dedicated end of acquisition and max count error flags with interrupt capability
11. One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
12. Compatible with proximity, touchkey, linear and rotary touch sensor implementation

### 40.2.2 How to use this driver

1. Enable the TSC interface clock using \_\_TSC\_CLK\_ENABLE() macro.
2. GPIO pins configuration
  - Enable the clock for the TSC GPIOs using \_\_GPIOx\_CLK\_ENABLE() macro.
  - Configure the TSC pins used as sampling IOs in alternate function output Open-Drain mode, and TSC pins used as channel/shield IOs in alternate function output Push-Pull mode using HAL\_GPIO\_Init() function.
  - Configure the alternate function on all the TSC pins using HAL\_xxxx() function.
3. Interrupts configuration
  - Configure the NVIC (if the interrupt model is used) using HAL\_xxx() function.
4. TSC configuration
  - Configure all TSC parameters and used TSC IOs using HAL\_TSC\_Init() function.

### Acquisition sequence

- Discharge all IOs using HAL\_TSC\_IODischarge() function.
- Wait a certain time allowing a good discharge of all capacitors. This delay depends of the sampling capacitor and electrodes design.
- Select the channel IOs to be acquired using HAL\_TSC\_IOConfig() function.
- Launch the acquisition using either HAL\_TSC\_Start() or HAL\_TSC\_Start\_IT() function. If the synchronized mode is selected, the acquisition will start as soon as the signal is received on the synchro pin.
- Wait the end of acquisition using either HAL\_TSC\_PollForAcquisition() or HAL\_TSC\_GetState() function or using WFI instruction for example.
- Check the group acquisition status using HAL\_TSC\_GroupGetStatus() function.
- Read the acquisition value using HAL\_TSC\_GroupGetValue() function.

#### 40.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the TSC.
- De-initialize the TSC.
- [\*\*HAL\\_TSC\\_Init\(\)\*\*](#)
- [\*\*HAL\\_TSC\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_TSC\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_TSC\\_MspDelInit\(\)\*\*](#)

#### 40.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure TSC IOs
- Discharge TSC IOs
- [\*\*HAL\\_TSC\\_IOConfig\(\)\*\*](#)
- [\*\*HAL\\_TSC\\_IODischarge\(\)\*\*](#)

#### 40.2.5 State functions

This subsection provides functions allowing to

- Get TSC state.

- Poll for acquisition completed.
- Handles TSC interrupt request.
- `HAL_TSC_GetState()`
- `HAL_TSC_PollForAcquisition()`
- `HAL_TSC_IRQHandler()`

#### 40.2.6 HAL\_TSC\_Init

Function Name	<code>HAL_StatusTypeDef HAL_TSC_Init ( TSC_HandleTypeDef * htsc)</code>
Function Description	Initializes the TSC peripheral according to the specified parameters in the <code>TSC_InitTypeDef</code> structure.
Parameters	<ul style="list-style-type: none"> <li>• <code>htsc</code> : TSC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 40.2.7 HAL\_TSC\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_TSC_DeInit ( TSC_HandleTypeDef * htsc)</code>
Function Description	Deinitializes the TSC peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <code>htsc</code> : TSC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 40.2.8 HAL\_TSC\_MspInit

Function Name	<code>void HAL_TSC_MspInit ( TSC_HandleTypeDef * htsc)</code>
Function Description	Initializes the TSC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <code>htsc</code> : pointer to a <code>TSC_HandleTypeDef</code> structure that contains the configuration information for the specified TSC.</li> </ul>

---

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

#### 40.2.9 HAL\_TSC\_MspDeInit

Function Name	<b>void HAL_TSC_MspDeInit ( <i>TSC_HandleTypeDef</i> * htsc)</b>
Function Description	Deinitializes the TSC MSP.
Parameters	<ul style="list-style-type: none"><li><b>htsc</b> : pointer to a <i>TSC_HandleTypeDef</i> structure that contains the configuration information for the specified TSC.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

#### 40.2.10 HAL\_TSC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TSC_Start ( <i>TSC_HandleTypeDef</i> * htsc)</b>
Function Description	Starts the acquisition.
Parameters	<ul style="list-style-type: none"><li><b>htsc</b> : pointer to a <i>TSC_HandleTypeDef</i> structure that contains the configuration information for the specified TSC.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

#### 40.2.11 HAL\_TSC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TSC_Start_IT ( <i>TSC_HandleTypeDef</i> * htsc)</b>
Function Description	Enables the interrupt and starts the acquisition.

---

Parameters	<ul style="list-style-type: none"> <li>• <b>htsc</b> : pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 40.2.12 HAL\_TSC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TSC_Stop ( <i>TSC_HandleTypeDef * htsc</i> )</b>
Function Description	Stops the acquisition previously launched in polling mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc</b> : pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 40.2.13 HAL\_TSC\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TSC_Stop_IT ( <i>TSC_HandleTypeDef * htsc</i> )</b>
Function Description	Stops the acquisition previously launched in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc</b> : pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 40.2.14 HAL\_TSC\_GroupGetStatus

Function Name	<b>TSC_GroupStatusTypeDef HAL_TSC_GroupGetStatus (</b>
---------------	--------------------------------------------------------

***TSC\_HandleTypeDef \* htsc, uint32\_t gx\_index)***

Function Description	Gets the acquisition status for a group.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc</b> : pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> <li>• <b>gx_index</b> : Index of the group</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Group status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**40.2.15 HAL\_TSC\_GroupGetValue**

Function Name	<b>uint32_t HAL_TSC_GroupGetValue ( <i>TSC_HandleTypeDef * htsc, uint32_t gx_index)</i></b>
Function Description	Gets the acquisition measure for a group.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc</b> : pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> <li>• <b>gx_index</b> : Index of the group</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Acquisition measure</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**40.2.16 HAL\_TSC\_IOConfig**

Function Name	<b>HAL_StatusTypeDef HAL_TSC_IOConfig ( <i>TSC_HandleTypeDef * htsc, TSC_IOConfigTypeDef * config)</i></b>
Function Description	Configures TSC IOs.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc</b> : pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> <li>• <b>config</b> : pointer to the configuration structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 40.2.17 HAL\_TSC\_IODischarge

Function Name	<b>HAL_StatusTypeDef HAL_TSC_IODischarge (</b> <b>TSC_HandleTypeDef * htsc, uint32_t choice)</b>
Function Description	Discharge TSC IOs.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc</b> : pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> <li>• <b>choice</b> : enable or disable</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 40.2.18 HAL\_TSC\_GetState

Function Name	<b>HAL_TSC_StateTypeDef HAL_TSC_GetState (</b> <b>TSC_HandleTypeDef * htsc)</b>
Function Description	Return the TSC state.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc</b> : pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 40.2.19 HAL\_TSC\_PollForAcquisition

Function Name	<b>HAL_StatusTypeDef HAL_TSC_PollForAcquisition (</b> <b>TSC_HandleTypeDef * htsc)</b>
Function Description	Start acquisition and wait until completion.
Parameters	<ul style="list-style-type: none"> <li>• <b>htsc</b> : pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• There is no need of a timeout parameter as the max count error is already managed by the TSC peripheral.</li> </ul>

#### 40.2.20 HAL\_TSC\_IRQHandler

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

#### 40.2.21 HAL\_TSC\_ConvCpltCallback

Function Name	<b>void HAL_TSC_ConvCpltCallback ( <i>TSC_HandleTypeDef</i> * htsc)</b>
Function Description	Acquisition completed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htsc</b> : pointer to a <i>TSC_HandleTypeDef</i> structure that contains the configuration information for the specified TSC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 40.2.22 HAL\_TSC\_ErrorCallback

Function Name	<b>void HAL_TSC_ErrorCallback ( <i>TSC_HandleTypeDef</i> * htsc)</b>
Function Description	Error callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htsc</b> : pointer to a <i>TSC_HandleTypeDef</i> structure that contains the configuration information for the specified TSC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

Notes

- None.

## 40.3 TSC Firmware driver defines

### 40.3.1 TSC

TSC

**TSC Acquisition mode**

TSC\_ACQ\_MODE\_NORMAL

TSC\_ACQ\_MODE\_SYNCHRO

IS\_TSC\_ACQ\_MODE

**TSC Charge Transfer Pulse High**

TSC\_CTPH\_1CYCLE

TSC\_CTPH\_2CYCLES

TSC\_CTPH\_3CYCLES

TSC\_CTPH\_4CYCLES

TSC\_CTPH\_5CYCLES

TSC\_CTPH\_6CYCLES

TSC\_CTPH\_7CYCLES

TSC\_CTPH\_8CYCLES

TSC\_CTPH\_9CYCLES

TSC\_CTPH\_10CYCLES

TSC\_CTPH\_11CYCLES

TSC\_CTPH\_12CYCLES

TSC\_CTPH\_13CYCLES

TSC\_CTPH\_14CYCLES

TSC\_CTPH\_15CYCLES

TSC\_CTPH\_16CYCLES

IS\_TSC\_CTPH

**TSC Charge Transfer Pulse Low**

TSC\_CTPL\_1CYCLE

TSC\_CTPL\_2CYCLES

TSC\_CTPL\_3CYCLES

TSC\_CTPL\_4CYCLES

TSC\_CTPL\_5CYCLES

TSC\_CTPL\_6CYCLES  
TSC\_CTPL\_7CYCLES  
TSC\_CTPL\_8CYCLES  
TSC\_CTPL\_9CYCLES  
TSC\_CTPL\_10CYCLES  
TSC\_CTPL\_11CYCLES  
TSC\_CTPL\_12CYCLES  
TSC\_CTPL\_13CYCLES  
TSC\_CTPL\_14CYCLES  
TSC\_CTPL\_15CYCLES  
TSC\_CTPL\_16CYCLES  
IS\_TSC\_CTPL

**TSC Exported Macros**

`_HAL_TSC_RESET_HANDLE_STATE`

**Description:**

- Reset TSC handle state.

**Parameters:**

- `_HANDLE_`: TSC handle.

**Return value:**

- None:

`_HAL_TSC_ENABLE`

**Description:**

- Enable the TSC peripheral.

**Parameters:**

- `_HANDLE_`: TSC handle

**Return value:**

- None:

`_HAL_TSC_DISABLE`

**Description:**

- Disable the TSC peripheral.

**Parameters:**

- `_HANDLE_`: TSC handle

**Return value:**

- None:

`_HAL_TSC_START_ACQ`

**Description:**

- Start acquisition.

**Parameters:**

- `_HANDLE_`: TSC handle

**Return value:**

- None:

`__HAL_TSC_STOP_ACQ`

**Description:**

- Stop acquisition.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None:

`__HAL_TSC_SET_IODEF_OUTPPLLOW`

**Description:**

- Set IO default mode to output push-pull low.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None:

`__HAL_TSC_SET_IODEF_INFLOAT`

**Description:**

- Set IO default mode to input floating.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None:

`__HAL_TSC_SET_SYNC_POL_FALL`

**Description:**

- Set synchronization polarity to falling edge.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None:

`__HAL_TSC_SET_SYNC_POL_RISE_HIGH`

**Description:**

- Set synchronization polarity to rising edge and high level.

**Parameters:**

- `__HANDLE__`: TSC handle

**Return value:**

- None:

`__HAL_TSC_ENABLE_IT`

**Description:**

- Enable TSC interrupt.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__INTERRUPT__`: TSC interrupt

**Return value:**

- None:

`__HAL_TSC_DISABLE_IT`

- Disable TSC interrupt.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__INTERRUPT__`: TSC interrupt

**Return value:**

- None:

`__HAL_TSC_GET_IT_SOURCE`

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

**Parameters:**

- `__HANDLE__`: TSC Handle
- `__INTERRUPT__`: TSC interrupt

**Return value:**

- SET: or RESET

`__HAL_TSC_GET_FLAG`

- Get the selected TSC's flag status.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__FLAG__`: TSC flag

**Return value:**

- SET: or RESET

`__HAL_TSC_CLEAR_FLAG`

- Clear the TSC's pending flag.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__FLAG__`: TSC flag

**Return value:**

- None:

`__HAL_TSC_ENABLE_HYSTERESIS`

- Enable schmitt trigger hysteresis on a group of IOs.

**Parameters:**

- `__HANDLE__`: TSC handle

- `__GX_IOY_MASK__`: IOs mask

**Description:**

- Disable schmitt trigger hysteresis on a group of IOs.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

**Description:**

- None:

**Description:**

- Open analog switch on a group of IOs.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

**Description:**

- None:

**Description:**

- Close analog switch on a group of IOs.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

**Description:**

- None:

**Description:**

- Enable a group of IOs in channel mode.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

**Description:**

- None:

**Description:**

- Disable a group of channel IOs.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

`__HAL_TSC_ENABLE_SAMPLING`

**Return value:**

- None:

**Description:**

- Enable a group of IOs in sampling mode.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

**Return value:**

- None:

`__HAL_TSC_DISABLE_SAMPLING`

**Description:**

- Disable a group of sampling IOs.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

**Return value:**

- None:

`__HAL_TSC_ENABLE_GROUP`

**Description:**

- Enable acquisition groups.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_MASK__`: Groups mask

**Return value:**

- None:

`__HAL_TSC_DISABLE_GROUP`

**Description:**

- Disable acquisition groups.

**Parameters:**

- `__HANDLE__`: TSC handle
- `__GX_MASK__`: Groups mask

**Return value:**

- None:

`__HAL_TSC_GET_GROUP_STATUS`

**Description:**

- Gets acquisition group status.

**Parameters:**

- `__HANDLE__`: TSC Handle
- `__GX_INDEX__`: Group index

**Return value:**

- SET: or RESET

***TSC Flags Definition***

TSC\_FLAG\_EOA

TSC\_FLAG\_MCE

***TSC groups definition***

TSC\_NB\_OF\_GROUPS

TSC\_GROUP1

TSC\_GROUP2

TSC\_GROUP3

TSC\_GROUP4

TSC\_GROUP5

TSC\_GROUP6

TSC\_GROUP7

TSC\_GROUP8

TSC\_ALL\_GROUPS

TSC\_GROUP1\_IDX

TSC\_GROUP2\_IDX

TSC\_GROUP3\_IDX

TSC\_GROUP4\_IDX

TSC\_GROUP5\_IDX

TSC\_GROUP6\_IDX

TSC\_GROUP7\_IDX

TSC\_GROUP8\_IDX

IS\_GROUP\_INDEX

TSC\_GROUP1\_IO1

TSC\_GROUP1\_IO2

TSC\_GROUP1\_IO3

TSC\_GROUP1\_IO4

TSC\_GROUP1\_ALL\_IOS

TSC\_GROUP2\_IO1

TSC\_GROUP2\_IO2

TSC\_GROUP2\_IO3

TSC\_GROUP2\_IO4

TSC\_GROUP2\_ALL\_IOS

TSC\_GROUP3\_IO1

TSC\_GROUP3\_IO2

TSC\_GROUP3\_IO3

TSC\_GROUP3\_IO4  
TSC\_GROUP3\_ALL\_IOS  
TSC\_GROUP4\_IO1  
TSC\_GROUP4\_IO2  
TSC\_GROUP4\_IO3  
TSC\_GROUP4\_IO4  
TSC\_GROUP4\_ALL\_IOS  
TSC\_GROUP5\_IO1  
TSC\_GROUP5\_IO2  
TSC\_GROUP5\_IO3  
TSC\_GROUP5\_IO4  
TSC\_GROUP5\_ALL\_IOS  
TSC\_GROUP6\_IO1  
TSC\_GROUP6\_IO2  
TSC\_GROUP6\_IO3  
TSC\_GROUP6\_IO4  
TSC\_GROUP6\_ALL\_IOS  
TSC\_GROUP7\_IO1  
TSC\_GROUP7\_IO2  
TSC\_GROUP7\_IO3  
TSC\_GROUP7\_IO4  
TSC\_GROUP7\_ALL\_IOS  
TSC\_GROUP8\_IO1  
TSC\_GROUP8\_IO2  
TSC\_GROUP8\_IO3  
TSC\_GROUP8\_IO4  
TSC\_GROUP8\_ALL\_IOS  
TSC\_ALL\_GROUPS\_ALL\_IOS

***TSC interrupts definition***

TSC\_IT\_EOA  
TSC\_IT\_MCE  
IS\_TSC\_MCE\_IT

***TSC I/O default mode definition***

TSC\_IODEF\_OUT\_PP\_LOW  
TSC\_IODEF\_IN\_FLOAT  
IS\_TSC\_IODEF

***TSC I/O mode definition***

TSC\_IOMODE\_UNUSED  
TSC\_IOMODE\_CHANNEL  
TSC\_IOMODE\_SHIELD  
TSC\_IOMODE\_SAMPLING  
IS\_TSC\_IOMODE

***TSC Max Count Value definition***

TSC\_MCV\_255  
TSC\_MCV\_511  
TSC\_MCV\_1023  
TSC\_MCV\_2047  
TSC\_MCV\_4095  
TSC\_MCV\_8191  
TSC\_MCV\_16383  
IS\_TSC\_MCV

***TSC Pulse Generator prescaler definition***

TSC\_PG\_PRESC\_DIV1  
TSC\_PG\_PRESC\_DIV2  
TSC\_PG\_PRESC\_DIV4  
TSC\_PG\_PRESC\_DIV8  
TSC\_PG\_PRESC\_DIV16  
TSC\_PG\_PRESC\_DIV32  
TSC\_PG\_PRESC\_DIV64  
TSC\_PG\_PRESC\_DIV128  
IS\_TSC\_PG\_PRESC

***TSC Spread Spectrum***

IS\_TSC\_SS  
IS\_TSC\_SSD

***TSC Spread spectrum prescaler definition***

TSC\_SS\_PRESC\_DIV1  
TSC\_SS\_PRESC\_DIV2  
IS\_TSC\_SS\_PRESC

***TSC Synchronization pin polarity***

TSC\_SYNC\_POL\_FALL  
TSC\_SYNC\_POL\_RISE\_HIGH  
IS\_TSC\_SYNC\_POL

# 41 HAL UART Generic Driver

## 41.1 UART Firmware driver registers structures

### 41.1.1 UART\_InitTypeDef

**UART\_InitTypeDef** is defined in the stm32f0xx\_hal\_uart.h

#### Data Fields

- **uint32\_t BaudRate**
- **uint32\_t WordLength**
- **uint32\_t StopBits**
- **uint32\_t Parity**
- **uint32\_t Mode**
- **uint32\_t HwFlowCtl**
- **uint32\_t OverSampling**
- **uint32\_t OneBitSampling**

#### Field Documentation

- **uint32\_t UART\_InitTypeDef::BaudRate** This member configures the UART communication baud rate. The baud rate register is computed using the following formula:
  - If oversampling is 16 or in LIN mode (LIN mode not available on F030xx devices), Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate)))
  - If oversampling is 8, Baud Rate Register[15:4] = ((2 \* PCLKx) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 \* PCLKx) / ((huart->Init.BaudRate)))[3:0]) >> 1
- **uint32\_t UART\_InitTypeDef::WordLength** Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [\*\*UARTEx\\_Word\\_Length\*\*](#)
- **uint32\_t UART\_InitTypeDef::StopBits** Specifies the number of stop bits transmitted. This parameter can be a value of [\*\*UART\\_Stop\\_Bits\*\*](#)
- **uint32\_t UART\_InitTypeDef::Parity** Specifies the parity mode. This parameter can be a value of [\*\*UART\\_Parity\*\*](#)  
**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\*\*](#)
- **uint32\_t UART\_InitTypeDef::OneBitSampling** Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of [\*\*UART\\_OneBit\\_Sampling\*\*](#).

### 41.1.2 **UART\_AdvFeatureInitTypeDef**

**UART\_AdvFeatureInitTypeDef** is defined in the stm32f0xx\_hal\_uart.h

#### Data Fields

- **uint32\_t AdvFeatureInit**
- **uint32\_t TxPinLevelInvert**
- **uint32\_t RxPinLevelInvert**
- **uint32\_t DataInvert**
- **uint32\_t Swap**
- **uint32\_t OverrunDisable**
- **uint32\_t DMADisableonRxError**
- **uint32\_t AutoBaudRateEnable**
- **uint32\_t AutoBaudRateMode**
- **uint32\_t MSBFirst**

#### Field Documentation

- **uint32\_t UART\_AdvFeatureInitTypeDef::AdvFeatureInit** Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of [\*\*UART\\_Advanced\\_Features\\_Initialization\\_Type\*\*](#)
- **uint32\_t UART\_AdvFeatureInitTypeDef::TxPinLevelInvert** Specifies whether the TX pin active level is inverted. This parameter can be a value of [\*\*UART\\_Tx\\_Inv\*\*](#)
- **uint32\_t UART\_AdvFeatureInitTypeDef::RxPinLevelInvert** Specifies whether the RX pin active level is inverted. This parameter can be a value of [\*\*UART\\_Rx\\_Inv\*\*](#)
- **uint32\_t UART\_AdvFeatureInitTypeDef::DataInvert** Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of [\*\*UART\\_Data\\_Inv\*\*](#)
- **uint32\_t UART\_AdvFeatureInitTypeDef::Swap** Specifies whether TX and RX pins are swapped. This parameter can be a value of [\*\*UART\\_Rx\\_Tx\\_Swap\*\*](#)
- **uint32\_t UART\_AdvFeatureInitTypeDef::OverrunDisable** Specifies whether the reception overrun detection is disabled. This parameter can be a value of [\*\*UART\\_Overrun\\_Disable\*\*](#)
- **uint32\_t UART\_AdvFeatureInitTypeDef::DMADisableonRxError** Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of [\*\*UART\\_DMA\\_Disable\\_on\\_Rx\\_Error\*\*](#)
- **uint32\_t UART\_AdvFeatureInitTypeDef::AutoBaudRateEnable** Specifies whether auto Baud rate detection is enabled. This parameter can be a value of [\*\*UART\\_AutoBaudRate\\_Enable\*\*](#)
- **uint32\_t UART\_AdvFeatureInitTypeDef::AutoBaudRateMode** If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of [\*\*UARTEx\\_AutoBaud\\_Rate\\_Mode\*\*](#)
- **uint32\_t UART\_AdvFeatureInitTypeDef::MSBFirst** Specifies whether MSB is sent first on UART line. This parameter can be a value of [\*\*UART\\_MSB\\_First\*\*](#)

### 41.1.3 **UART\_WakeUpTypeDef**

**UART\_WakeUpTypeDef** is defined in the stm32f0xx\_hal\_uart.h

#### Data Fields

- **uint32\_t WakeUpEvent**
- **uint16\_t AddressLength**

- ***uint8\_t Address***

#### Field Documentation

- ***uint32\_t UART\_WakeUpTypeDef::WakeUpEvent*** Specifies which event will activate the Wakeup from Stop mode flag (WUF). This parameter can be a value of [\*\*UART\\_WakeUp\\_from\\_Stop\\_Selection\*\*](#). If set to **UART\_WAKEUP\_ON\_ADDRESS**, the two other fields below must be filled up.
- ***uint16\_t UART\_WakeUpTypeDef::AddressLength*** Specifies whether the address is 4 or 7-bit long. This parameter can be a value of [\*\*UART\\_WakeUp\\_Address\\_Length\*\*](#)
- ***uint8\_t UART\_WakeUpTypeDef::Address*** UART/USART node address (7-bit long max)

### 41.1.4 **UART\_HandleTypeDef**

**UART\_HandleTypeDef** is defined in the `stm32f0xx_hal_uart.h`

#### Data Fields

- ***USART\_TypeDef \* Instance***
- ***UART\_InitTypeDef Init***
- ***UART\_AdvFeatureInitTypeDef AdvancedInit***
- ***uint8\_t \* pTxBuffPtr***
- ***uint16\_t TxXferSize***
- ***uint16\_t TxXferCount***
- ***uint8\_t \* pRxBuffPtr***
- ***uint16\_t RxXferSize***
- ***uint16\_t RxXferCount***
- ***uint16\_t Mask***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***HAL\_LockTypeDef Lock***
- ***HAL\_UART\_StateTypeDef State***
- ***HAL\_UART\_ErrorTypeDef ErrorCode***

#### Field Documentation

- ***USART\_TypeDef\* UART\_HandleTypeDef::Instance***
- ***UART\_InitTypeDef UART\_HandleTypeDef::Init***
- ***UART\_AdvFeatureInitTypeDef UART\_HandleTypeDef::AdvancedInit***
- ***uint8\_t\* UART\_HandleTypeDef::pTxBuffPtr***
- ***uint16\_t UART\_HandleTypeDef::TxXferSize***
- ***uint16\_t UART\_HandleTypeDef::TxXferCount***
- ***uint8\_t\* UART\_HandleTypeDef::pRxBuffPtr***
- ***uint16\_t UART\_HandleTypeDef::RxXferSize***
- ***uint16\_t UART\_HandleTypeDef::RxXferCount***
- ***uint16\_t UART\_HandleTypeDef::Mask***
- ***DMA\_HandleTypeDef\* UART\_HandleTypeDef::hdmatx***
- ***DMA\_HandleTypeDef\* UART\_HandleTypeDef::hdmarx***
- ***HAL\_LockTypeDef UART\_HandleTypeDef::Lock***
- ***HAL\_UART\_StateTypeDef UART\_HandleTypeDef::State***
- ***HAL\_UART\_ErrorTypeDef UART\_HandleTypeDef::ErrorCode***

## 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 these UART pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (`HAL_UART_Transmit_IT()` and `HAL_UART_Receive_IT()` APIs):
    - Configure the USARTx interrupt priority.
    - Enable the NVIC USART IRQ handle.
  - d. DMA Configuration if you need to use DMA process (`HAL_UART_Transmit_DMA()` and `HAL_UART_Receive_DMA()` APIs):
    - Declare a DMA handle structure for the Tx/Rx 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.
3. Program the Baud Rate, Word Length , Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the `huart Init` structure.
4. If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...) in the `huart AdvancedInit` structure.
5. For the UART asynchronous mode, initialize the UART registers by calling the `HAL_UART_Init()` API.
6. For the UART Half duplex mode, initialize the UART registers by calling the `HAL_HalfDuplex_Init()` API.
7. For the UART Multiprocessor mode, initialize the UART registers by calling the `HAL_MultiProcessor_Init()` API.
8. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the `HAL_RS485Ex_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()`, `HAL_HalfDuplex_Init()`, `HAL_MultiProcessor_Init()`, also 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 half transfer HAL\_UART\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxHalfCpltCallback
- At transmission end of transfer HAL\_UART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_UART\_Receive\_IT()
- At reception end of half transfer HAL\_UART\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxHalfCpltCallback
- At reception end of transfer HAL\_UART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxCpltCallback
- In case of transfer Error, HAL\_UART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_UART\_ErrorCallback

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL\_UART\_Transmit\_DMA()
- At transmission end of half transfer HAL\_UART\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxHalfCpltCallback
- At transmission end of transfer HAL\_UART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_UART\_Receive\_DMA()
- At reception end of half transfer HAL\_UART\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxHalfCpltCallback
- At reception end of transfer HAL\_UART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxCpltCallback
- In case of transfer Error, HAL\_UART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_UART\_ErrorCallback
- Pause the DMA Transfer using HAL\_UART\_DMAPause()
- Resume the DMA Transfer using HAL\_UART\_DMAResume()
- Stop the DMA Transfer using HAL\_UART\_DMAStop()

### UART HAL driver macros list

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

- `_HAL_UART_ENABLE`: Enable the UART peripheral
- `_HAL_UART_DISABLE`: Disable the UART peripheral
- `_HAL_UART_GET_FLAG` : Check whether the specified UART flag is set or not
- `_HAL_UART_CLEAR_FLAG` : Clear the specified UART pending flag
- `_HAL_UART_ENABLE_IT`: Enable the specified UART interrupt
- `_HAL_UART_DISABLE_IT`: Disable the specified UART interrupt



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

### 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 the below table.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

Table 23: UART frame formats

M1, M0 bits	PCE bit	UART frame
00	0	SB   8 bit data   STB
00	1	SB   7 bit data   PB   STB
01	0	SB   9 bit data   STB
01	1	SB   8 bit data   PB   STB
10	0	SB   7 bit data   STB
10	1	SB   6 bit data   PB   STB

The HAL\_UART\_Init(), HAL\_HalfDuplex\_Init() and HAL\_MultiProcessor\_Init() API follow respectively the UART asynchronous, UART Half duplex and multiprocessor configuration procedures (details for the procedures are available in reference manual).

- [\*\*HAL\\_UART\\_Init\(\)\*\*](#)
- [\*\*HAL\\_HalfDuplex\\_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 mode of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA, These 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()
  - UART\_Transmit\_IT()
  - UART\_Receive\_IT()
4. Non Blocking mode APIs 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\_TxHalfCpltCallback()
  - HAL\_UART\_TxCpltCallback()
  - HAL\_UART\_RxHalfCpltCallback()
  - HAL\_UART\_RxCpltCallback()
  - HAL\_UART\_ErrorCallback()



In the Half duplex communication, it is forbidden to run the transmit and receive process in parallel, the UART state HAL\_UART\_STATE\_BUSY\_TX\_RX can't be useful.

- [`HAL\_UART\_Transmit\(\)`](#)
- [`HAL\_UART\_Receive\(\)`](#)
- [`HAL\_UART\_Transmit\_IT\(\)`](#)
- [`HAL\_UART\_Receive\_IT\(\)`](#)
- [`HAL\_UART\_Transmit\_DMA\(\)`](#)
- [`HAL\_UART\_Receive\_DMA\(\)`](#)
- [`HAL\_UART\_DMAPause\(\)`](#)
- [`HAL\_UART\_DMAResume\(\)`](#)
- [`HAL\_UART\_DMADMAStop\(\)`](#)
- [`HAL\_UART\_TxCpltCallback\(\)`](#)
- [`HAL\_UART\_TxHalfCpltCallback\(\)`](#)
- [`HAL\_UART\_RxCpltCallback\(\)`](#)
- [`HAL\_UART\_RxHalfCpltCallback\(\)`](#)
- [`HAL\_UART\_ErrorCallback\(\)`](#)

#### 41.2.4 Peripheral Control functions

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

- `HAL_UART_GetState()` API is helpful to check in run-time the state of the UART peripheral.
- `HAL_MultiProcessor_EnableMuteMode()` API enables mute mode
- `HAL_MultiProcessor_DisableMuteMode()` API disables mute mode
- `HAL_MultiProcessor_EnterMuteMode()` API enters mute mode
- `HAL_MultiProcessor_EnableMuteMode()` API enables mute mode
- `HAL_UART_EnableStopMode()` API enables the UART to wake up the MCU from stop mode
- `HAL_UART_DisableStopMode()` API disables the above functionality
- `UART_SetConfig()` API configures the UART peripheral
- `UART_AdvFeatureConfig()` API optionally configures the UART advanced features
- `UART_CheckIdleState()` API ensures that TEACK and/or REACK are set after initialization
- `UART_Wakeup_AddressConfig()` API configures the wake-up from stop mode parameters
- `HAL_HalfDuplex_EnableTransmitter()` API disables receiver and enables transmitter
- `HAL_HalfDuplex_EnableReceiver()` API disables transmitter and enables receiver
- [`HAL\_MultiProcessor\_EnableMuteMode\(\)`](#)
- [`HAL\_MultiProcessor\_DisableMuteMode\(\)`](#)
- [`HAL\_MultiProcessor\_EnterMuteMode\(\)`](#)
- [`HAL\_HalfDuplex\_EnableTransmitter\(\)`](#)
- [`HAL\_HalfDuplex\_EnableReceiver\(\)`](#)

#### 41.2.5 `HAL_UART_Init`

Function Name	<code>HAL_StatusTypeDef HAL_UART_Init ( <a href="#">UART_HandleTypeDef</a> * huart)</code>
Function Description	Initializes the UART mode according to the specified parameters in the <code>UART_InitTypeDef</code> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <code>huart</code> : uart handle</li> </ul>

---

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

#### 41.2.6 HAL\_HalfDuplex\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_Init (</b> <i>UART_HandleTypeDef * huart)</i>
Function Description	Initializes the half-duplex mode according to the specified parameters in the <i>UART_InitTypeDef</i> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.7 HAL\_MultiProcessor\_Init

Function Name	<b>HAL_StatusTypeDef HAL_MultiProcessor_Init (</b> <i>UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)</i>
Function Description	Initializes the multiprocessor mode according to the specified parameters in the <i>UART_InitTypeDef</i> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : UART handle</li> <li>• <b>Address</b> : UART node address (4-, 6-, 7- or 8-bit long)</li> <li>• <b>WakeUpMethod</b> : specifies the UART wakeup method. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>UART_WAKEUPMETHOD_IDLELINE</b> WakeUp by an idle line detection</li> <li>- <b>UART_WAKEUPMETHOD_ADDRESSMARK</b> WakeUp by an address mark</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function.</li> <li>• If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses</li> </ul>

detection, the API  
HAL\_MultiProcessorEx\_AddressLength\_Set() must be called  
after HAL\_MultiProcessor\_Init().

#### 41.2.8 HAL\_UART\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_UART_DelInit (</b> <b>  UART_HandleTypeDef * huart)</b>
Function Description	Deinitializes the UART peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.9 HAL\_UART\_MspInit

Function Name	<b>void HAL_UART_MspInit (</b> <b>  UART_HandleTypeDef * huart)</b>
Function Description	UART MSP Init.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.10 HAL\_UART\_MspDelInit

Function Name	<b>void HAL_UART_MspDelInit (</b> <b>  UART_HandleTypeDef * huart)</b>
Function Description	UART MSP DelInit.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

## Notes

- None.

#### 41.2.11 HAL\_UART\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit (</b> <b>UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size,</b> <b>uint32_t Timeout)</b>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.12 HAL\_UART\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive (</b> <b>UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size,</b> <b>uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</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> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.13 HAL\_UART\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit_IT (</b> <b>UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.14 HAL\_UART\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive_IT (</b> <b>UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.15 HAL\_UART\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit_DMA (</b> <b>UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.16 HAL\_UART\_Receive\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_UART_Receive_DMA (</code> <code>UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</code>
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</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>• <b>HAL status</b></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>

#### 41.2.17 HAL\_UART\_DMAPause

Function Name	<code>HAL_StatusTypeDef HAL_UART_DMAPause (</code> <code>UART_HandleTypeDef * huart)</code>
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : UART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.18 HAL\_UART\_DMAResume

Function Name	<code>HAL_StatusTypeDef HAL_UART_DMAResume (</code> <code>UART_HandleTypeDef * huart)</code>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : UART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.19 HAL\_UART\_DMASStop

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMASStop (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : UART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.20 HAL\_UART\_TxCpltCallback

Function Name	<b>void HAL_UART_TxCpltCallback (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.21 HAL\_UART\_TxHalfCpltCallback

Function Name	<b>void HAL_UART_TxHalfCpltCallback (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : UART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.22 HAL\_UART\_RxCpltCallback

Function Name	<code>void HAL_UART_RxCpltCallback ( <i>UART_HandleTypeDef</i> *  <b>huart</b>)</code>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.23 HAL\_UART\_RxHalfCpltCallback

Function Name	<code>void HAL_UART_RxHalfCpltCallback ( <i>UART_HandleTypeDef</i> *  <b>huart</b>)</code>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : UART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.24 HAL\_UART\_ErrorCallback

Function Name	<code>void HAL_UART_ErrorCallback ( <i>UART_HandleTypeDef</i> *  <b>huart</b>)</code>
Function Description	UART error callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.25 HAL\_MultiProcessor\_EnableMuteMode

Function Name	<code>HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (     <i>UART_HandleTypeDef</i> * huart)</code>
Function Description	Enable UART in mute mode (doesn't mean UART enters mute mode; to enter mute mode, <code>HAL_MultiProcessor_EnterMuteMode()</code> API must be called)
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.26 HAL\_MultiProcessor\_DisableMuteMode

Function Name	<code>HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (     <i>UART_HandleTypeDef</i> * huart)</code>
Function Description	Disable UART mute mode (doesn't mean it actually wakes up the software, as it may not have been in mute mode at this very moment).
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.27 HAL\_MultiProcessor\_EnterMuteMode

Function Name	<code>void HAL_MultiProcessor_EnterMuteMode (     <i>UART_HandleTypeDef</i> * huart)</code>
Function Description	Enter UART mute mode (means UART actually enters mute mode).

---

Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.28 HAL\_HalfDuplex\_EnableTransmitter

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Enables the UART transmitter and disables the UART receiver.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : UART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.29 HAL\_HalfDuplex\_EnableReceiver

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Enables the UART receiver and disables the UART transmitter.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : UART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.30 HAL\_UART\_GetState

Function Name	<b>HAL_UART_StateTypeDef HAL_UART_GetState (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	return the UART state

---

Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.31 HAL\_UART\_GetError

Function Name	<b>uint32_t HAL_UART_GetError ( <a href="#">UART_HandleTypeDef</a> * <b>huart</b>)</b>
Function Description	Return the UART error code.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <a href="#">UART_HandleTypeDef</a> structure that contains the configuration information for the specified UART.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>UART Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 41.3 UART Firmware driver defines

#### 41.3.1 UART

UART

**UART Advanced Feature Initialization Type**

UART\_ADVFEATURE\_NO\_INIT

UART\_ADVFEATURE\_TXINVERT\_INIT

UART\_ADVFEATURE\_RXINVERT\_INIT

UART\_ADVFEATURE\_DATAINVERT\_INIT

UART\_ADVFEATURE\_SWAP\_INIT

UART\_ADVFEATURE\_RXOVERRUNDISABLE\_INIT

UART\_ADVFEATURE\_DMADISABLEONERROR\_INIT

UART\_ADVFEATURE\_AUTOBAUDRATE\_INIT

UART\_ADVFEATURE\_MSBFIRST\_INIT

IS\_UART\_ADVFEATURE\_INIT

**UART Advanced Feature Auto BaudRate Enable**

UART\_ADVFEATURE\_AUTOBAUDRATE\_DISABLE

UART\_ADVFEATURE\_AUTOBAUDRATE\_ENABLE  
IS\_UART\_ADVFEATURE\_AUTOBAUDRATE  
**UART Driver Enable Assertion Time LSB Position In CR1 Register**  
UART\_CR1\_DEAT\_ADDRESS\_LSB\_POS  
**UART Driver Enable DeAssertion Time LSB Position In CR1 Register**  
UART\_CR1\_DEDT\_ADDRESS\_LSB\_POS  
**UART Address-matching LSB Position In CR2 Register**  
UART\_CR2\_ADDRESS\_LSB\_POS  
**UART Advanced Feature Binary Data Inversion**  
UART\_ADVFEATURE\_DATAINV\_DISABLE  
UART\_ADVFEATURE\_DATAINV\_ENABLE  
IS\_UART\_ADVFEATURE\_DATAINV  
**UART Advanced Feature DMA Disable On Rx Error**  
UART\_ADVFEATURE\_DMA\_ENABLEONRXERROR  
UART\_ADVFEATURE\_DMA\_DISABLEONRXERROR  
IS\_UART\_ADVFEATURE\_DMAONRXERROR  
**UART DMA Rx**  
UART\_DMA\_RX\_DISABLE  
UART\_DMA\_RX\_ENABLE  
IS\_UART\_DMA\_RX  
**UART DMA Tx**  
UART\_DMA\_TX\_DISABLE  
UART\_DMA\_TX\_ENABLE  
IS\_UART\_DMA\_TX  
**UART DriverEnable Polarity**  
UART\_DE\_POLARITY\_HIGH  
UART\_DE\_POLARITY\_LOW  
IS\_UART\_DE\_POLARITY  
**UART Exported Macros**

`_HAL_UART_RESET_HANDLE_STA` **Description:**  
TE                   • Reset UART handle state.  
**Parameters:**  
• `_HANDLE_`: UART handle.  
**Return value:**  
• None:  
**Description:**  
• Checks whether the specified UART flag is

`_HAL_UART_GET_FLAG`

set or not.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be `UARTx` where `x`: 1, 2, 3 or 4 to select the USART or UART peripheral (datasheet: up to four USART/UARTs)
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `UART_FLAG_RXACK`: Receive enable acknowledge flag
  - `UART_FLAG_TEACK`: Transmit enable acknowledge flag
  - `UART_FLAG_WUF`: Wake up from stop mode flag (not available on F030xx devices)
  - `UART_FLAG_RWU`: Receiver wake up flag (not available on F030xx devices)
  - `UART_FLAG_SBKF`: Send Break flag
  - `UART_FLAG_CMF`: Character match flag
  - `UART_FLAG_BUSY`: Busy flag
  - `UART_FLAG_ABRF`: Auto Baud rate detection flag
  - `UART_FLAG_ABRE`: Auto Baud rate detection error flag
  - `UART_FLAG_EOBF`: End of block flag (not available on F030xx devices)
  - `UART_FLAG_RTOF`: Receiver timeout flag
  - `UART_FLAG_CTS`: CTS Change flag
  - `UART_FLAG_LBD`: LIN Break detection flag (not available on F030xx devices)
  - `UART_FLAG_TXE`: Transmit data register empty flag
  - `UART_FLAG_TC`: Transmission Complete flag
  - `UART_FLAG_RXNE`: Receive data register not empty flag
  - `UART_FLAG_IDLE`: Idle Line detection flag
  - `UART_FLAG_ORE`: OverRun Error flag
  - `UART_FLAG_NE`: Noise Error flag
  - `UART_FLAG_FE`: Framing Error flag
  - `UART_FLAG_PE`: Parity Error flag

**Return value:**

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

[\\_\\_HAL\\_UART\\_ENABLE\\_IT](#)**Description:**

- Enables the specified UART interrupt.

**Parameters:**

- HANDLE: specifies the UART Handle. This parameter can be USARTx where x: 1, 2, 3 or 4 to select the USART or UART peripheral. (datasheet: up to four USART/UARTs)
- INTERRUPT: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART\_IT\_WUF: Wakeup from stop mode interrupt (not available on F030xx devices)
  - UART\_IT\_CM: Character match interrupt
  - UART\_IT\_CTS: CTS change interrupt
  - UART\_IT\_LBD: LIN Break detection interrupt (not available on F030xx devices)
  - UART\_IT\_TXE: Transmit Data Register empty interrupt
  - UART\_IT\_TC: Transmission complete interrupt
  - UART\_IT\_RXNE: Receive Data register not empty interrupt
  - UART\_IT\_IDLE: Idle line detection interrupt
  - UART\_IT\_PE: Parity Error interrupt
  - UART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

[\\_\\_HAL\\_UART\\_DISABLE\\_IT](#)**Description:**

- Disables the specified UART interrupt.

**Parameters:**

- HANDLE: specifies the UART Handle. This parameter can be USARTx where x: 1, 2, 3 or 4 to select the USART or UART peripheral. (datasheet: up to four USART/UARTs)
- INTERRUPT: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - UART\_IT\_WUF: Wakeup from stop mode interrupt (not available on F030xx devices)
  - UART\_IT\_CM: Character match interrupt
  - UART\_IT\_CTS: CTS change interrupt

- UART\_IT\_LBD: LIN Break detection interrupt (not available on F030xx devices)
- UART\_IT\_TXE: Transmit Data Register empty interrupt
- UART\_IT\_TC: Transmission complete interrupt
- UART\_IT\_RXNE: Receive Data register not empty interrupt
- UART\_IT\_IDLE: Idle line detection interrupt
- UART\_IT\_PE: Parity Error interrupt
- UART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None:

**\_HAL\_UART\_GET\_IT**

- Checks whether the specified UART interrupt has occurred or not.

**Parameters:**

- \_HANDLE\_: specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3 or 4 to select the USART or UART peripheral. (datasheet: up to four USART/UARTs)
- \_IT\_: specifies the UART interrupt to check. This parameter can be one of the following values:
  - UART\_IT\_WUF: Wakeup from stop mode interrupt (not available on F030xx devices)
  - UART\_IT\_CM: Character match interrupt
  - UART\_IT\_CTS: CTS change interrupt
  - UART\_IT\_LBD: LIN Break detection interrupt (not available on F030xx devices)
  - 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\_ORE: OverRun Error interrupt
  - UART\_IT\_NE: Noise Error interrupt
  - UART\_IT\_FE: Framing Error interrupt
  - UART\_IT\_PE: Parity Error interrupt

[\\_\\_HAL\\_UART\\_GET\\_IT\\_SOURCE](#)**Return value:**

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

**Description:**

- Checks whether the specified UART interrupt source is enabled.

**Parameters:**

- \_\_HANDLE\_\_: specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3 or 4 to select the USART or UART peripheral. (datasheet: up to four USART/UARTs)
- \_\_IT\_\_: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - UART\_IT\_WUF: Wakeup from stop mode interrupt (not available on F030xx devices)
  - UART\_IT\_CM: Character match interrupt
  - UART\_IT\_CTS: CTS change interrupt
  - UART\_IT\_LBD: LIN Break detection interrupt (not available on F030xx devices)
  - 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\_ORE: OverRun Error interrupt
  - UART\_IT\_NE: Noise Error interrupt
  - UART\_IT\_FE: Framing Error interrupt
  - UART\_IT\_PE: Parity Error interrupt

**Return value:**

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

[\\_\\_HAL\\_UART\\_CLEAR\\_IT](#)**Description:**

- Clears the specified UART ISR flag, in setting the proper ICR register flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3 or 4 to select the USART or UART peripheral. (datasheet: up to four USART/UARTs)
- \_\_IT\_CLEAR\_\_: specifies the interrupt clear register flag that needs to be set to clear the

corresponding interrupt This parameter can be one of the following values:

- UART\_CLEAR\_PEF: Parity Error Clear Flag
- UART\_CLEAR\_FEF: Framing Error Clear Flag
- UART\_CLEAR\_NEF: Noise detected Clear Flag
- UART\_CLEAR\_OREF: OverRun Error Clear Flag
- UART\_CLEAR\_IDLEF: IDLE line detected Clear Flag
- UART\_CLEAR\_TCF: Transmission Complete Clear Flag
- UART\_CLEAR\_LBDF: LIN Break Detection Clear Flag (not available on F030xx devices)
- UART\_CLEAR\_CTSF: CTS Interrupt Clear Flag
- UART\_CLEAR\_RTOF: Receiver Time Out Clear Flag
- UART\_CLEAR\_EOBF: End Of Block Clear Flag (not available on F030xx devices)
- UART\_CLEAR\_CMF: Character Match Clear Flag
- UART\_CLEAR\_WUF: Wake Up from stop mode Clear Flag (not available on F030xx devices)

#### Return value:

- None:

### \_HAL\_UART\_SEND\_REQ

- Set a specific UART request flag.

#### Parameters:

- \_HANDLE\_: specifies the UART Handle. This parameter can be USARTx where x: 1, 2, 3 or 4 to select the USART or UART peripheral. (datasheet: up to four USART/UARTs)
- \_REQ\_: specifies the request flag to set. This parameter can be one of the following values:
  - UART\_AUTOBAUD\_REQUEST: Auto-Baud Rate Request
  - UART\_SENDBREAK\_REQUEST: Send Break Request
  - UART\_MUTE\_MODE\_REQUEST: Mute Mode Request
  - UART\_RXDATA\_FLUSH\_REQUEST: Receive Data flush Request
  - UART\_TXDATA\_FLUSH\_REQUEST:

Transmit data flush Request (not available on F030xx devices)

**Return value:**

- None:

`__HAL_UART_ENABLE`

**Description:**

- Enable UART.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. The Handle Instance can be `UARTx` where `x: 1, 2, 3, 4 or 5` to select the UART peripheral

**Return value:**

- None:

`__HAL_UART_DISABLE`

**Description:**

- Disable UART.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. The Handle Instance can be `UARTx` where `x: 1, 2, 3, 4 or 5` to select the UART peripheral

**Return value:**

- None:

***UARTEx Status Flags***

`UART_FLAG_RXACK`

`UART_FLAG_TEACK`

`UART_FLAG_WUF`

`UART_FLAG_RWU`

`UART_FLAG_SBKF`

`UART_FLAG_CMF`

`UART_FLAG_BUSY`

`UART_FLAG_ABRF`

`UART_FLAG_ABRE`

`UART_FLAG_EOBF`

`UART_FLAG_RTOF`

`UART_FLAG_CTS`

`UART_FLAG_CTSIF`

`UART_FLAG_LBDF`

`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 Half Duplex Selection***

UART\_HALF\_DUPLEX\_DISABLE  
UART\_HALF\_DUPLEX\_ENABLE  
IS\_UART\_HALF\_DUPLEX

***UART Hardware Flow Control***

UART\_HWCONTROL\_NONE  
UART\_HWCONTROL\_RTS  
UART\_HWCONTROL\_CTS  
UART\_HWCONTROL\_RTS\_CTS  
IS\_UART\_HARDWARE\_FLOW\_CONTROL

***UART Interruptions Flag Mask***

UART\_IT\_MASK

***UARTEx Interrupts Definition***

UART\_IT\_PE  
UART\_IT\_TXE  
UART\_IT\_TC  
UART\_IT\_RXNE  
UART\_IT\_IDLE  
UART\_IT\_LBD  
UART\_IT\_CTS  
UART\_IT\_CM  
UART\_IT\_WUF

***UART IT***

UART\_IT\_ERR  
UART\_IT\_ORE  
UART\_IT\_NE  
UART\_IT\_FE

***UARTEx Interruption Clear Flags***

UART\_CLEAR\_PEF      Parity Error Clear Flag

---

UART_CLEAR_FEF	Framing Error Clear Flag
UART_CLEAR_NEF	Noise detected Clear Flag
UART_CLEAR_OREF	OverRun Error Clear Flag
UART_CLEAR_IDLEF	IDLE line detected Clear Flag
UART_CLEAR_TCF	Transmission Complete Clear Flag
UART_CLEAR_LBDF	LIN Break Detection Clear Flag (not available on F030xx devices)
UART_CLEAR_CTSF	CTS Interrupt Clear Flag
UART_CLEAR_RTOF	Receiver Time Out Clear Flag
UART_CLEAR_EOBF	End Of Block Clear Flag
UART_CLEAR_CMF	Character Match Clear Flag
UART_CLEAR_WUF	Wake Up from stop mode Clear Flag

***UART Transfer Mode***

UART\_MODE\_RX  
UART\_MODE\_TX  
UART\_MODE\_TX\_RX  
IS\_UART\_MODE

***UART Advanced Feature MSB First***

UART\_ADVFEATURE\_MSBFIRST\_DISABLE  
UART\_ADVFEATURE\_MSBFIRST\_ENABLE  
IS\_UART\_ADVFEATURE\_MSBFIRST

***UART Advanced Feature Mute Mode Enable***

UART\_ADVFEATURE\_MUTEMODE\_DISABLE  
UART\_ADVFEATURE\_MUTEMODE\_ENABLE  
IS\_UART\_MUTE\_MODE

***UART One Bit Sampling Method***

UART\_ONEBIT\_SAMPLING\_DISABLED  
UART\_ONEBIT\_SAMPLING\_ENABLED  
IS\_UART\_ONEBIT\_SAMPLING

***UART One Bit sampling***

UART\_ONE\_BIT\_SAMPLE\_DISABLED  
UART\_ONE\_BIT\_SAMPLE\_ENABLED  
IS\_UART\_ONEBIT\_SAMPLE

***UART Advanced Feature Overrun Disable***

UART\_ADVFEATURE\_OVERRUN\_ENABLE  
UART\_ADVFEATURE\_OVERRUN\_DISABLE  
IS\_UART\_OVERRUN

***UART Over Sampling***

UART\_OVERSAMPLING\_16

UART\_OVERSAMPLING\_8

IS\_UART\_OVERSAMPLING

**UART Parity**

UART\_PARITY\_NONE

UART\_PARITY EVEN

UART\_PARITY ODD

IS\_UART\_PARITY

**UART Private Constants**

HAL\_UART\_TXDMA\_TIMEOUTVALUE

UART\_CR1\_FIELDS

**UART Private Macros**

\_DIV\_SAMPLING8

**Description:**

- BRR division operation to set BRR register in 8-bit oversampling mode.

**Parameters:**

- \_PCLK\_: UART clock
- \_BAUD\_: Baud rate set by the user

**Return value:**

- Division: result

\_DIV\_SAMPLING16

**Description:**

- BRR division operation to set BRR register in 16-bit oversampling mode.

**Parameters:**

- \_PCLK\_: UART clock
- \_BAUD\_: Baud rate set by the user

**Return value:**

- Division: result

IS\_UART\_BAUDRATE

**Description:**

- Check UART Baud rate.

**Parameters:**

- BAUDRATE: Baudrate specified by the user The maximum Baud Rate is derived from the maximum clock on F0 (i.e. 48 MHz) divided by the smallest oversampling used on the USART (i.e. 8)

**Return value:**

- Test: result (TRUE or FALSE).

IS\_UART\_ASSERTIONTIME

**Description:**

- Check UART assertion time.

**Parameters:**

- TIME: 5-bit value assertion time

**Return value:**

- Test: result (TRUE or FALSE).

IS\_UART\_DEASSERTIONTIME

**Description:**

- Check UART deassertion time.

**Parameters:**

- TIME: 5-bit value deassertion time

**Return value:**

- Test: result (TRUE or FALSE).

**UART Receiver TimeOut**

UART\_RECEIVER\_TIMEOUT\_DISABLE

UART\_RECEIVER\_TIMEOUT\_ENABLE

IS\_UART\_RECEIVER\_TIMEOUT

**UARTEx Request Parameters**

UART\_AUTOBAUD\_REQUEST Auto-Baud Rate Request

UART\_SENDBREAK\_REQUEST Send Break Request

UART\_MUTE\_MODE\_REQUEST Mute Mode Request

UART\_RXDATA\_FLUSH\_REQUEST Receive Data flush Request

UART\_TXDATA\_FLUSH\_REQUEST Transmit data flush Request

IS\_UART\_REQUEST\_PARAMETER

**UART Advanced Feature RX Pin Active Level Inversion**

UART\_ADVFEATURE\_RXINV\_DISABLE

UART\_ADVFEATURE\_RXINV\_ENABLE

IS\_UART\_ADVFEATURE\_RXINV

**UART Advanced Feature RX TX Pins Swap**

UART\_ADVFEATURE\_SWAP\_DISABLE

UART\_ADVFEATURE\_SWAP\_ENABLE

IS\_UART\_ADVFEATURE\_SWAP

**UART State**

UART\_STATE\_DISABLE

UART\_STATE\_ENABLE

IS\_UART\_STATE

**UART Number of Stop Bits**

UART\_STOPBITS\_1

UART\_STOPBITS\_2

UART\_STOPBITS\_1\_5

IS\_UART\_STOPBITS

**UARTE<sub>x</sub> Advanced Feature Stop Mode Enable**

UART\_ADVFEATURE\_STOPMODE\_DISABLE

UART\_ADVFEATURE\_STOPMODE\_ENABLE

IS\_UART\_ADVFEATURE\_STOPMODE

**UART polling-based communications time-out value**

HAL\_UART\_TIMEOUT\_VALUE

**UART Advanced Feature TX Pin Active Level Inversion**

UART\_ADVFEATURE\_TXINV\_DISABLE

UART\_ADVFEATURE\_TXINV\_ENABLE

IS\_UART\_ADVFEATURE\_TXINV

**UART WakeUp Address Length**

UART\_ADDRESS\_DETECT\_4B

UART\_ADDRESS\_DETECT\_7B

IS\_UART\_ADDRESSLENGTH\_DETECT

**UART WakeUp From Stop Selection**

UART\_WAKEUP\_ON\_ADDRESS

UART\_WAKEUP\_ON\_STARTBIT

UART\_WAKEUP\_ON\_READDATA\_NONEMPTY

IS\_UART\_WAKEUP\_SELECTION

**UART WakeUp Methods**

UART\_WAKEUPMETHOD\_IDLELINE

UART\_WAKEUPMETHOD\_ADDRESSMARK

IS\_UART\_WAKEUPMETHOD

## 42 HAL UART Extension Driver

### 42.1 UARTEEx Firmware driver API description

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

#### 42.1.1 Initialization and Configuration functions

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

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length (Fixed to 8-bits only for LIN mode)
  - 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 the following table:

M1M0 bits	PCE bit	UART frame	SB	8-bit data	STB	SB	7-bit data	PB	STB	SB	9-bit data	STB	SB	8-bit data	PB	STB	SB	7-bit data	STB	SB	6-bit data	PB	STB						
---	00	0	SB	8-bit data	STB	---	---	SB	7-bit data	PB	STB	---	SB	9-bit data	STB	---	SB	8-bit data	PB	STB	+	SB	7-bit data	STB	+	SB	6-bit data	PB	STB
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	+	---	---	---	+	---	---	+	---	
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL\_LIN\_Init() and HAL\_RS485Ex\_Init() APIs follows respectively the LIN and the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

- [\*\*HAL\\_RS485Ex\\_Init\(\)\*\*](#)
- [\*\*HAL\\_LIN\\_Init\(\)\*\*](#)

#### 42.1.2 IO operation function

This subsection provides functions allowing to manage the UART interrupts and to handle Wake up interrupt call-back.

1. Non-Blocking mode API with Interrupt is :
  - `HAL_UART_IRQHandler()`
2. Callback provided in No\_Blocking mode:
  - `HAL_UART_WakeupCallback()`
  - `HAL_UART_IRQHandler()`
  - `HAL_UART_WakeupCallback()`

#### 42.1.3 Peripheral Control function

This subsection provides extended functions allowing to control the UART.

- `HAL_MultiProcessorEx_AddressLength_Set()` API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- `HAL_UARTEx_StopModeWakeUpSourceConfig()` API sets Wakeup from Stop mode interrupt flag selection
- `HAL_UARTEx_EnableStopMode()` API allows the UART to wake up the MCU from Stop mode as long as UART clock is HSI or LSE
- `HAL_UARTEx_DisableStopMode()` API disables the above feature
- `HAL_LIN_SendBreak()` API transmits the break characters
- `HAL_UARTEx_StopModeWakeUpSourceConfig()`
- `HAL_UARTEx_EnableStopMode()`
- `HAL_UARTEx_DisableStopMode()`
- `HAL_MultiProcessorEx_AddressLength_Set()`
- `HAL_LIN_SendBreak()`

#### 42.1.4 HAL\_RS485Ex\_Init

Function Name	<code>HAL_StatusTypeDef HAL_RS485Ex_Init (</code> <code>UART_HandleTypeDef * huart, uint32_t UART_DEPolarity,</code> <code>uint32_t UART_DEAffirmationTime, uint32_t</code> <code>UART_DEDeaffirmationTime)</code>
Function Description	Initializes the RS485 Driver enable feature according to the specified parameters in the <code>UART_InitTypeDef</code> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle</li> <li>• <b>UART_DEPolarity</b> : select the driver enable polarity This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <code>UART_DE_POLARITY_HIGH</code> DE signal is active high</li> <li>- <code>UART_DE_POLARITY_LOW</code> DE signal is active low</li> </ul> </li> <li>• <b>UART_DEAffirmationTime</b> : Driver Enable assertion time 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate)</li> <li>• <b>UART_DEDeaffirmationTime</b> : Driver Enable deassertion time 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of</li> </ul>

the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).

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

#### 42.1.5 HAL\_LIN\_Init

Function Name	<b>HAL_StatusTypeDef HAL_LIN_Init ( <i>UART_HandleTypeDef</i> * <i>huart</i>, <i>uint32_t BreakDetectLength</i>)</b>
Function Description	Initializes the LIN mode according to the specified parameters in the <i>UART_InitTypeDef</i> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle</li> <li>• <b>BreakDetectLength</b> : specifies the LIN break detection length. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>UART_LINBREAKDETECTLENGTH_10B</i></b> 10-bit break detection</li> <li>– <b><i>UART_LINBREAKDETECTLENGTH_11B</i></b> 11-bit break detection</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 42.1.6 HAL\_UART\_IRQHandler

Function Name	<b>void HAL_UART_IRQHandler ( <i>UART_HandleTypeDef</i> * <i>huart</i>)</b>
Function Description	This function handles UART interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 42.1.7 HAL\_UART\_WakeupCallback

Function Name	<code>void HAL_UART_WakeupCallback ( <i>UART_HandleTypeDef</i> * huart)</code>
Function Description	UART wakeup from Stop mode callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 42.1.8 HAL\_UARTEx\_StopModeWakeUpSourceConfig

Function Name	<code>HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig ( <i>UART_HandleTypeDef</i> * huart, <i>UART_WakeUpTypeDef</i> WakeUpSelection)</code>
Function Description	Set Wakeup from Stop mode interrupt flag selection.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : uart handle,</li> <li>• <b>WakeUpSelection</b> : address match, Start Bit detection or RXNE bit status. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <i>UART_WAKEUP_ON_ADDRESS</i></li> <li>- <i>UART_WAKEUP_ON_STARTBIT</i></li> <li>- <i>UART_WAKEUP_ON_RXNE</i></li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 42.1.9 HAL\_UARTEx\_EnableStopMode

Function Name	<code>HAL_StatusTypeDef HAL_UARTEx_EnableStopMode ( <i>UART_HandleTypeDef</i> * huart)</code>
Function Description	Enable UART Stop Mode The UART is able to wake up the MCU from Stop mode as long as UART clock is HSI or LSE.

---

Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 42.1.10 HAL\_UARTEx\_DisableStopMode

Function Name	<b>HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Disable UART Stop Mode.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : uart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 42.1.11 HAL\_MultiProcessorEx\_AddressLength\_Set

Function Name	<b>HAL_StatusTypeDef</b> <b>HAL_MultiProcessorEx_AddressLength_Set (</b> <b><i>UART_HandleTypeDef</i> * huart, uint32_t AddressLength)</b>
Function Description	By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses detection.
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 42.1.12 HAL\_LIN\_SendBreak

Function Name	<b>HAL_StatusTypeDef HAL_LIN_SendBreak (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Transmits break characters.

---

Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 42.2 UARTE Firmware driver defines

### 42.2.1 UARTE

UARTE

***UARTE Advanced Feature AutoBaud Rate Mode***

UART\_ADVFEATURE\_AUTOBAUDRATE\_ONSTARTBIT  
 UART\_ADVFEATURE\_AUTOBAUDRATE\_ONFALLINGEDGE  
 UART\_ADVFEATURE\_AUTOBAUDRATE\_ON0X7FFFRAME  
 UART\_ADVFEATURE\_AUTOBAUDRATE\_ON0X55FRAME  
 IS\_UART\_ADVFEATURE\_AUTOBAUDRATEMODE

***UARTE Exported Macros***

<code>_HAL_UART_GETCLOCKSOURCE</code>	<b>Description:</b> <ul style="list-style-type: none"> <li>• Reports the UART clock source.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: specifies the UART Handle</li> <li>• <code>_CLOCKSOURCE_</code>: output variable</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• UART: clocking source, written in <code>_CLOCKSOURCE_</code>.</li> </ul>
<code>_HAL_UART_MASK_COMPUTATION</code>	<b>Description:</b> <ul style="list-style-type: none"> <li>• Computes the UART mask to apply to retrieve the received data according to the word length and to the parity bits activation.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: specifies the UART Handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• none:</li> </ul>

***UARTE Local Interconnection Network mode***

UART\_LIN\_DISABLE  
 UART\_LIN\_ENABLE  
 IS\_UART\_LIN

***UARTE LIN Break Detection***

UART\_LINBREAKDETECTLENGTH\_10B

UART\_LINBREAKDETECTLENGTH\_11B

IS\_UART\_LIN\_BREAK\_DETECT\_LENGTH

***UARTEx Word Length***

UART\_WORDLENGTH\_7B

UART\_WORDLENGTH\_8B

UART\_WORDLENGTH\_9B

IS\_UART\_WORD\_LENGTH

## 43 HAL USART Generic Driver

### 43.1 USART Firmware driver registers structures

#### 43.1.1 USART\_InitTypeDef

**USART\_InitTypeDef** is defined in the stm32f0xx\_hal\_usart.h

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t CLKLastBit*

##### Field Documentation

- ***uint32\_t USART\_InitTypeDef::BaudRate*** This member configures the Usart communication baud rate. The baud rate is computed using the following formula:  
Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate)))
- ***uint32\_t USART\_InitTypeDef::WordLength*** Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of **USARTEx\_Word\_Length**
- ***uint32\_t USART\_InitTypeDef::StopBits*** Specifies the number of stop bits transmitted. This parameter can be a value of **USART\_Stop\_Bits**
- ***uint32\_t USART\_InitTypeDef::Parity*** Specifies the parity mode. This parameter can be a value of **USART\_Parity**

**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**

#### 43.1.2 USART\_HandleTypeDef

**USART\_HandleTypeDef** is defined in the stm32f0xx\_hal\_usart.h

##### Data Fields

- ***USART\_TypeDef \* Instance***
- ***USART\_InitTypeDef Init***

- `uint8_t * pTxBuffPtr`
- `uint16_t TxXferSize`
- `uint16_t TxXferCount`
- `uint8_t * pRxBuffPtr`
- `uint16_t RxXferSize`
- `uint16_t RxXferCount`
- `uint16_t Mask`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `HAL_LockTypeDef Lock`
- `HAL_USART_StateTypeDef State`
- `HAL_USART_ErrorTypeDef ErrorCode`

#### Field Documentation

- `USART_TypeDef* USART_HandleTypeDef::Instance` USART registers base address
- `USART_InitTypeDef USART_HandleTypeDef::Init` USART communication parameters
- `uint8_t* USART_HandleTypeDef::pTxBuffPtr` Pointer to USART Tx transfer Buffer
- `uint16_t USART_HandleTypeDef::TxXferSize` USART Tx Transfer size
- `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
- `uint16_t USART_HandleTypeDef::RxXferCount` USART Rx Transfer Counter
- `uint16_t USART_HandleTypeDef::Mask` USART Rx RDR register mask
- `DMA_HandleTypeDef* USART_HandleTypeDef::hdmatx` USART Tx DMA Handle parameters
- `DMA_HandleTypeDef* USART_HandleTypeDef::hdmarx` USART Rx DMA Handle parameters
- `HAL_LockTypeDef USART_HandleTypeDef::Lock` Locking object
- `HAL_USART_StateTypeDef USART_HandleTypeDef::State` USART communication state
- `HAL_USART_ErrorTypeDef USART_HandleTypeDef::ErrorCode` USART Error code

## 43.2 USART Firmware driver API description

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

### 43.2.1 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 USARTTx interface clock.
  - b. USART pins configuration:
    - Enable the clock for the USART GPIOs.
    - Configure these USART pins as alternate function pull-up.

- c. NVIC configuration if you need to use interrupt process (HAL\_USART\_Transmit\_IT(), HAL\_USART\_Receive\_IT() and HAL\_USART\_TransmitReceive\_IT() APIs):
    - Configure the USARTx interrupt priority.
    - Enable the NVIC USART IRQ handle.
  - d. DMA Configuration if you need to use DMA process (HAL\_USART\_Transmit\_DMA() HAL\_USART\_Receive\_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.
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 half transfer HAL\_USART\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxHalfCpltCallback
- At transmission end of transfer HAL\_USART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_USART\_Receive\_IT()
- At reception end of half transfer HAL\_USART\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_RxHalfCpltCallback
- At reception end of transfer HAL\_USART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_RxCpltCallback
- In case of transfer Error, HAL\_USART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_USART\_ErrorCallback

## DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL\_USART\_Transmit\_DMA()
- At transmission end of half transfer HAL\_USART\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxHalfCpltCallback
- At transmission end of transfer HAL\_USART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_USART\_Receive\_DMA()
- At reception end of half transfer HAL\_USART\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_RxHalfCpltCallback
- At reception end of transfer HAL\_USART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_RxCpltCallback
- In case of transfer Error, HAL\_USART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_USART\_ErrorCallback
- Pause the DMA Transfer using HAL\_USART\_DMAPause()
- Resume the DMA Transfer using HAL\_USART\_DMAResume()
- Stop the DMA Transfer using HAL\_USART\_DMAStop()

## USART HAL driver macros list

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

- \_\_HAL\_USART\_ENABLE: Enable the USART peripheral
- \_\_HAL\_USART\_DISABLE: Disable the USART peripheral
- \_\_HAL\_USART\_GET\_FLAG : Check whether the specified USART flag is set or not
- \_\_HAL\_USART\_CLEAR\_FLAG : Clear the specified USART pending flag
- \_\_HAL\_USART\_ENABLE\_IT: Enable the specified USART interrupt
- \_\_HAL\_USART\_DISABLE\_IT: Disable the specified USART interrupt



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

### 43.2.2 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) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit), the possible USART frame formats are as listed in the below table.

- USART polarity
- USART phase
- USART LastBit
- Receiver/transmitter modes

Table 24: 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
M1, M0 bits	PCE bit	USART frame
10	0	SB   7 bit data   STB
10	1	SB   6 bit data   PB   STB

The HAL\_USART\_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

- [\*\*HAL\\_USART\\_Init\(\)\*\*](#)
- [\*\*HAL\\_USART\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_USART\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_USART\\_MspDeInit\(\)\*\*](#)
- [\*\*HAL\\_USART\\_CheckIdleState\(\)\*\*](#)

### 43.2.3 IO operation functions

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

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - 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 USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_USART\_TxCpltCallback(), HAL\_USART\_RxCpltCallback() and HAL\_USART\_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL\_USART\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
  - HAL\_USART\_Transmit() in simplex mode
  - HAL\_USART\_Receive() in full duplex receive only
  - HAL\_USART\_TransmitReceive() in full duplex mode
3. Non Blocking mode APIs with Interrupt are :
  - HAL\_USART\_Transmit\_IT() in simplex mode
  - HAL\_USART\_Receive\_IT() in full duplex receive only

- HAL\_USART\_TransmitReceive\_IT() in full duplex mode
  - HAL\_USART\_IRQHandler()
4. Non Blocking mode functions with DMA are :
- HAL\_USART\_Transmit\_DMA() in simplex mode
  - HAL\_USART\_Receive\_DMA() in full duplex receive only
  - HAL\_USART\_TransmitReceive\_DMA() in full duplex mode
  - HAL\_USART\_DMAPause()
  - HAL\_USART\_DMAResume()
  - HAL\_USART\_DMAStop()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
- HAL\_USART\_TxCpltCallback()
  - HAL\_USART\_RxCpltCallback()
  - HAL\_USART\_TxHalfCpltCallback()
  - HAL\_USART\_RxHalfCpltCallback()
  - HAL\_USART\_ErrorCallback()
  - HAL\_USART\_TxRxCpltCallback()
  - ***HAL\_USART\_Transmit()***
  - ***HAL\_USART\_Receive()***
  - ***HAL\_USART\_TransmitReceive()***
  - ***HAL\_USART\_Transmit\_IT()***
  - ***HAL\_USART\_Receive\_IT()***
  - ***HAL\_USART\_TransmitReceive\_IT()***
  - ***HAL\_USART\_Transmit\_DMA()***
  - ***HAL\_USART\_Receive\_DMA()***
  - ***HAL\_USART\_TransmitReceive\_DMA()***
  - ***HAL\_USART\_DMAPause()***
  - ***HAL\_USART\_DMAResume()***
  - ***HAL\_USART\_DMAStop()***
  - ***HAL\_USART\_IRQHandler()***
  - ***HAL\_USART\_TxCpltCallback()***
  - ***HAL\_USART\_TxHalfCpltCallback()***
  - ***HAL\_USART\_RxCpltCallback()***
  - ***HAL\_USART\_RxHalfCpltCallback()***
  - ***HAL\_USART\_TxRxCpltCallback()***
  - ***HAL\_USART\_ErrorCallback()***

#### 43.2.4 Peripheral Control functions

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

- HAL\_USART\_GetState() API can be helpful to check in run-time the state of the USART peripheral.
- USART\_SetConfig() API is used to set the USART communication parameters.
- USART\_CheckIdleState() API ensures that TEACK and/or REACK bits are set after initialization
- ***HAL\_USART\_GetState()***
- ***HAL\_USART\_GetError()***

#### 43.2.5 HAL\_USART\_Init

---

Function Name	<b>HAL_StatusTypeDef HAL_USART_Init (</b> <b><i>USART_HandleTypeDef * husart</i></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> : usart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 43.2.6 HAL\_USART\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_USART_DelInit (</b> <b><i>USART_HandleTypeDef * husart</i></b> )
Function Description	Delinitializes the USART peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : usart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 43.2.7 HAL\_USART\_MspInit

Function Name	<b>void HAL_USART_MspInit (</b> <b><i>USART_HandleTypeDef * husart</i></b> )
Function Description	USART MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : usart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 43.2.8 HAL\_USART\_MspDelInit

Function Name	<b>void HAL_USART_MspDelInit ( <i>USART_HandleTypeDef</i> * husart)</b>
Function Description	USART MSP DelInit.
Parameters	<ul style="list-style-type: none"><li><b>husart</b> : usart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 43.2.9 HAL\_USART\_CheckIdleState

Function Name	<b>HAL_StatusTypeDef HAL_USART_CheckIdleState ( <i>USART_HandleTypeDef</i> * husart)</b>
Function Description	
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 43.2.10 HAL\_USART\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit ( <i>USART_HandleTypeDef</i> * husart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Simplex Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li><b>husart</b> : USART handle</li><li><b>pTxData</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	<b>HAL status</b>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 43.2.11 HAL\_USART\_Receive

---

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive (</b> <b>USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t</b> <b>Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : USART handle</li> <li>• <b>pRxData</b> : pointer to data buffer</li> <li>• <b>Size</b> : amount of data to be received</li> <li>• <b>Timeout</b> : : Timeout duration</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 43.2.12 HAL\_USART\_TransmitReceive

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive (</b> <b>USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t *</b> <b>pRxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Full-Duplex Send and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : USART handle</li> <li>• <b>pTxData</b> : pointer to TX data buffer</li> <li>• <b>pRxData</b> : pointer to RX data buffer</li> <li>• <b>Size</b> : amount of data to be sent (same amount to be received)</li> <li>• <b>Timeout</b> : : Timeout duration</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 43.2.13 HAL\_USART\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit_IT (</b> <b>USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t</b> <b>Size)</b>
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : USART handle</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>pTxData</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 43.2.14 HAL\_USART\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive_IT (</b> <b>USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t</b> <b>Size)</b>
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : usart handle</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 43.2.15 HAL\_USART\_TransmitReceive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (</b> <b>USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t *</b> <b>pRxData, uint16_t Size)</b>
Function Description	Full-Duplex Send and Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : USART handle</li> <li>• <b>pTxData</b> : pointer to TX data buffer</li> <li>• <b>pRxData</b> : pointer to RX data buffer</li> <li>• <b>Size</b> : amount of data to be sent (same amount to be received)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 43.2.16 HAL\_USART\_Transmit\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_Transmit_DMA (     <b>USART_HandleTypeDef</b> * <b>husart</b>, uint8_t * <b>pTxData</b>, uint16_t     <b>Size</b>)</code>
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : USART handle</li> <li>• <b>pTxData</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 43.2.17 HAL\_USART\_Receive\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_Receive_DMA (     <b>USART_HandleTypeDef</b> * <b>husart</b>, uint8_t * <b>pRxData</b>, uint16_t     <b>Size</b>)</code>
Function Description	Full-Duplex Receive an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : USART handle</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)</li> <li>• The USART DMA transmit channel must be configured in order to generate the clock for the slave.</li> </ul>

### 43.2.18 HAL\_USART\_TransmitReceive\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (     <b>USART_HandleTypeDef</b> * <b>husart</b>, uint8_t * <b>pTxData</b>, uint8_t *)</code>
---------------	---------------------------------------------------------------------------------------------------------------------------------------------------------

	<b>pRxData, uint16_t Size)</b>
Function Description	Full-Duplex Transmit Receive an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : usart handle</li> <li>• <b>pTxData</b> : pointer to TX data buffer</li> <li>• <b>pRxData</b> : pointer to RX data buffer</li> <li>• <b>Size</b> : amount of data to be received/sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the USART parity is enabled (PCE = 1) the data received contain the parity bit.</li> </ul>

### 43.2.19 HAL\_USART\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAPause (</b> <b><i>USART_HandleTypeDef</i> * husart)</b>
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 43.2.20 HAL\_USART\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAResume (</b> <b><i>USART_HandleTypeDef</i> * husart)</b>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 43.2.21 HAL\_USART\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAStop (</b> <b><i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 43.2.22 HAL\_USART\_IRQHandler

Function Name	<b>void HAL_USART_IRQHandler (</b> <b><i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	This function handles USART interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 43.2.23 HAL\_USART\_TxCpltCallback

Function Name	<b>void HAL_USART_TxCpltCallback (</b> <b><i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : usart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

### 43.2.24 HAL\_USART\_TxHalfCpltCallback

Function Name	<b>void HAL_USART_TxHalfCpltCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 43.2.25 HAL\_USART\_RxCpltCallback

Function Name	<b>void HAL_USART_RxCpltCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 43.2.26 HAL\_USART\_RxHalfCpltCallback

Function Name	<b>void HAL_USART_RxHalfCpltCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : usart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

**43.2.27 HAL\_USART\_TxRxCpltCallback**

Function Name	<b>void HAL_USART_TxRxCpltCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Tx/Rx Transfers completed callback for the non-blocking process.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : usart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

**43.2.28 HAL\_USART\_ErrorCallback**

Function Name	<b>void HAL_USART_ErrorCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	USART error callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : usart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

**43.2.29 HAL\_USART\_GetState**

Function Name	<b>HAL_USART_StateTypeDef HAL_USART_GetState ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	return the USART state
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 43.2.30 HAL\_USART\_GetError

Function Name	<code>uint32_t HAL_USART_GetError ( USART_HandleTypeDef * husart)</code>
Function Description	Return the USART error code.
Parameters	<ul style="list-style-type: none"> <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>• <b>USART Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 43.3 USART Firmware driver defines

### 43.3.1 USART

USART

***USART Clock***

`USART_CLOCK_DISABLED`

`USART_CLOCK_ENABLED`

`IS_USART_CLOCK`

***USART Clock Phase***

`USART_PHASE_1EDGE`

`USART_PHASE_2EDGE`

`IS_USART_PHASE`

***USART Clock Polarity***

`USART_POLARITY_LOW`

`USART_POLARITY_HIGH`

`IS_USART_POLARITY`

***USART Exported Macros***

`_HAL_USART_RESET_HANDLE_ST  
ATE`      **Description:**

- Reset USART handle state.

**Parameters:**

- `_HANDLE_`: USART handle.

**Return value:**

- None:

[\\_\\_HAL\\_USART\\_GET\\_FLAG](#)**Description:**

- Checks whether the specified USART flag is set or not.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the USART Handle
- [\\_\\_FLAG\\_\\_](#): specifies the flag to check. This parameter can be one of the following values:
  - [USART\\_FLAG\\_RXACK](#): Receive enable acknowledge flag
  - [USART\\_FLAG\\_TEACK](#): Transmit enable acknowledge flag
  - [USART\\_FLAG\\_BUSY](#): Busy flag
  - [USART\\_FLAG\\_CTS](#): CTS Change flag
  - [USART\\_FLAG\\_TXE](#): Transmit data register empty flag
  - [USART\\_FLAG\\_TC](#): Transmission Complete flag
  - [USART\\_FLAG\\_RXNE](#): Receive data register not empty flag
  - [USART\\_FLAG\\_IDLE](#): Idle Line detection flag
  - [USART\\_FLAG\\_ORE](#): OverRun Error flag
  - [USART\\_FLAG\\_NE](#): Noise Error flag
  - [USART\\_FLAG\\_FE](#): Framing Error flag
  - [USART\\_FLAG\\_PE](#): Parity Error flag

**Return value:**

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

[\\_\\_HAL\\_USART\\_ENABLE\\_IT](#)**Description:**

- Enables the specified USART interrupt.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the USART Handle
- [\\_\\_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:

[\\_\\_HAL\\_USART\\_DISABLE\\_IT](#)

- Disables the specified USART interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle.
- \_\_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:

[\\_\\_HAL\\_USART\\_GET\\_IT](#)

- Checks whether the specified USART interrupt has occurred or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle
- \_\_IT\_\_: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_ORE: OverRun Error interrupt
  - USART\_IT\_NE: Noise Error interrupt
  - USART\_IT\_FE: Framing Error interrupt
  - USART\_IT\_PE: Parity Error interrupt

**Return value:**

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

[\\_\\_HAL\\_USART\\_GET\\_IT\\_SOURCE](#)**Description:**

- Checks whether the specified USART interrupt source is enabled.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the USART Handle.
- [\\_\\_IT\\_\\_](#): specifies the USART interrupt source to check. This parameter can be one of the following values:
  - [USART\\_IT\\_TXE](#): Transmit Data Register empty interrupt
  - [USART\\_IT\\_TC](#): Transmission complete interrupt
  - [USART\\_IT\\_RXNE](#): Receive Data register not empty interrupt
  - [USART\\_IT\\_IDLE](#): Idle line detection interrupt
  - [USART\\_IT\\_ORE](#): OverRun Error interrupt
  - [USART\\_IT\\_NE](#): Noise Error interrupt
  - [USART\\_IT\\_FE](#): Framing Error interrupt
  - [USART\\_IT\\_PE](#): Parity Error interrupt

**Return value:**

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

[\\_\\_HAL\\_USART\\_CLEAR\\_IT](#)**Description:**

- Clears the specified USART ISR flag, in setting the proper ICR register flag.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the USART Handle.
- [\\_\\_IT\\_CLEAR\\_\\_](#): specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - [USART\\_CLEAR\\_PEF](#): Parity Error Clear Flag
  - [USART\\_CLEAR\\_FEF](#): Framing Error Clear Flag
  - [USART\\_CLEAR\\_NEF](#): Noise detected Clear Flag
  - [USART\\_CLEAR\\_OREF](#): OverRun Error Clear Flag
  - [USART\\_CLEAR\\_IDLEF](#): IDLE line detected Clear Flag
  - [USART\\_CLEAR\\_TCF](#): Transmission Complete Clear Flag
  - [USART\\_CLEAR\\_CTSF](#): CTS Interrupt Clear Flag

**Return value:**

- None:

**\_HAL\_USART\_SEND\_REQ****Description:**

- Set a specific USART request flag.

**Parameters:**

- HANDLE: specifies the USART Handle.
- REQ: specifies the request flag to set. This parameter can be one of the following values:
  - USART\_RXDATA\_FLUSH\_REQUEST : Receive Data flush Request
  - USART\_TXDATA\_FLUSH\_REQUEST: Transmit data flush Request

**Return value:**

- None:

**\_HAL\_USART\_ENABLE****Description:**

- Enable USART.

**Parameters:**

- HANDLE: specifies the USART Handle.

**Return value:**

- None:

**\_HAL\_USART\_DISABLE****Description:**

- Disable USART.

**Parameters:**

- HANDLE: specifies the USART Handle.

**Return value:**

- None:

***USART Flags***

USART\_FLAG\_RXACK

USART\_FLAG\_TEACK

USART\_FLAG\_BUSY

USART\_FLAG\_CTS

USART\_FLAG\_CTSIF

USART\_FLAG\_LBDF

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 interruptions flag mask***

USART\_IT\_MASK

***USART Interrupts Definition***

USART\_IT\_PE

USART\_IT\_TXE

USART\_IT\_TC

USART\_IT\_RXNE

USART\_IT\_IDLE

USART\_IT\_ERR

USART\_IT\_ORE

USART\_IT\_NE

USART\_IT\_FE

***USART Interruption Clear Flags***

USART\_CLEAR\_PEF      Parity Error Clear Flag

USART\_CLEAR\_FEF      Framing Error Clear Flag

USART\_CLEAR\_NEF      Noise detected Clear Flag

USART\_CLEAR\_OREF      OverRun Error Clear Flag

USART\_CLEAR\_IDLEF      IDLE line detected Clear Flag

USART\_CLEAR\_TCF      Transmission Complete Clear Flag

USART\_CLEAR\_CTSF      CTS Interrupt Clear Flag

***USART Last Bit***

USART\_LASTBIT\_DISABLE

USART\_LASTBIT\_ENABLE

IS\_USART\_LASTBIT

***USART Mode***

USART\_MODE\_RX

USART\_MODE\_TX

USART\_MODE\_TX\_RX

IS\_USART\_MODE

***USART Parity***

USART\_PARITY\_NONE

USART\_PARITY\_EVEN

USART\_PARITY\_ODD

IS\_USART\_PARITY

**USART Private Constants**

DUMMY\_DATA

TEACK\_RXACK\_TIMEOUT

USART\_TXDMA\_TIMEOUTVALUE

USART\_TIMEOUT\_VALUE

USART\_CR1\_FIELDS

USART\_CR2\_FIELDS

**USART Private Macros**

IS\_USART\_BAUDRATE **Description:**

- Check USART Baud rate.

**Parameters:**

- BAUDRATE: Baudrate specified by the user. The maximum Baud Rate is derived from the maximum clock on F0 (i.e. 48 MHz) divided by the smallest oversampling used on the USART (i.e. 8)

**Return value:**

- Test: result (TRUE or FALSE)

**USART Request Parameters**

USART\_RXDATA\_FLUSH\_REQUEST Receive Data flush Request

USART\_TXDATA\_FLUSH\_REQUEST Transmit data flush Request

IS\_USART\_REQUEST\_PARAMETER

**USART Number of Stop Bits**

USART\_STOPBITS\_1

USART\_STOPBITS\_2

USART\_STOPBITS\_1\_5

IS\_USART\_STOPBITS

## 44 HAL USART Extension Driver

### 44.1 USARTEx Firmware driver defines

#### 44.1.1 USARTEx

USARTEx

##### ***USARTEx Exported Macros***

`_HAL_USART_GETCLOCKSOURCE`

##### **Description:**

- Reports the USART clock source.

##### **Parameters:**

- `_HANDLE_`: specifies the USART Handle
- `_CLOCKSOURCE_`: output variable

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

- the: USART clocking source, written in `_CLOCKSOURCE_`.

`_HAL_USART_MASK_COMPUTATION`

##### **Description:**

- Reports the USART mask to apply to retrieve the received data according to the word length and to the parity bits activation.

##### **Parameters:**

- `_HANDLE_`: specifies the USART Handle

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

- none:

##### ***USARTEx Word Length***

`USART_WORDLENGTH_7B`

`USART_WORDLENGTH_8B`

`USART_WORDLENGTH_9B`

`IS_USART_WORD_LENGTH`

## 45 HAL WWDG Generic Driver

### 45.1 WWDG Firmware driver registers structures

#### 45.1.1 WWDG\_InitTypeDef

*WWDG\_InitTypeDef* is defined in the `stm32f0xx_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

#### 45.1.2 WWDG\_HandleTypeDefDef

*WWDG\_HandleTypeDefDef* is defined in the `stm32f0xx_hal_wwdg.h`

##### Data Fields

- *WWDG\_TypeDef \* Instance*
- *WWDG\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_WWDG\_StateTypeDef State*

##### Field Documentation

- *WWDG\_TypeDef\* WWDG\_HandleTypeDefDef::Instance* Register base address
- *WWDG\_InitTypeDef WWDG\_HandleTypeDefDef::Init* WWDG required parameters
- *HAL\_LockTypeDef WWDG\_HandleTypeDefDef::Lock* WWDG locking object
- *\_\_IO HAL\_WWDG\_StateTypeDef WWDG\_HandleTypeDefDef::State* WWDG communication state

### 45.2 WWDG Firmware driver API description

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

#### 45.2.1 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.
- WWdg clock (Hz) = PCLK / (4096 \* Prescaler)
- WWdg timeout (mS) = 1000 \* (T[5;0] + 1) / 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 \* (Counter-Window) / WWdg clock
  - max time (mS) = 1000 \* (Counter-0x40) / WWdg clock
- Min-max timeout value @48 MHz(PCLK): ~85,3us / ~5,46 ms.

#### 45.2.2 How to use this driver

- Enable WWdg APB1 clock using \_\_WWdg\_CLK\_ENABLE().
- Set the WWdg prescaler, refresh window and counter value using HAL\_WWDG\_Init() function.
- Start the WWdg using HAL\_WWDG\_Start() function. When the WWdg is enabled the counter value should be configured to a value greater than 0x40 to prevent generating an immediate reset.
- Optionally you can enable the Early Wakeup Interrupt (EWI) which is generated when the counter reaches 0x40, and then start the WWdg using HAL\_WWDG\_Start\_IT(). 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

#### 45.2.3 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\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_WWDG\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_WWDG\\_MspDelInit\(\)\*\*](#)
- [\*\*HAL\\_WWDG\\_WakeupCallback\(\)\*\*](#)

#### 45.2.4 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\(\)\*\*](#)

#### 45.2.5 Peripheral State functions

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

- [\*\*HAL\\_WWDG\\_GetState\(\)\*\*](#)

#### 45.2.6 HAL\_WWDG\_Init

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Init (</b> <b>WWDG_HandleTypeDef * hwdg)</b>
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>hwdg</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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 45.2.7 HAL\_WWDG\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_DelInit ( <i>WWDG_HandleTypeDef</i> * hwdg)</b>
Function Description	Deinitializes the WWDG peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hwdg</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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 45.2.8 HAL\_WWDG\_MspInit

Function Name	<b>void HAL_WWDG_MspInit ( <i>WWDG_HandleTypeDef</i> * hwdg)</b>
Function Description	Initializes the WWDG MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hwdg</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>• None.</li></ul>

#### 45.2.9 HAL\_WWDG\_MspDelInit

Function Name	<b>void HAL_WWDG_MspDelInit ( <i>WWDG_HandleTypeDef</i> * hwdg)</b>
Function Description	Deinitializes the WWDG MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hwdg</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>• None.</li></ul>

#### 45.2.10 HAL\_WWDG\_WakeupCallback

Function Name	<code>void HAL_WWDG_WakeupCallback ( <i>WWDG_HandleTypeDef</i> * hwdg)</code>
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hwdg</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>• None.</li></ul>

#### 45.2.11 HAL\_WWDG\_Start

Function Name	<code>HAL_StatusTypeDef HAL_WWDG_Start ( <i>WWDG_HandleTypeDef</i> * hwdg)</code>
Function Description	Starts the WWDG.
Parameters	<ul style="list-style-type: none"><li>• <b>hwdg</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>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 45.2.12 HAL\_WWDG\_Start\_IT

Function Name	<code>HAL_StatusTypeDef HAL_WWDG_Start_IT ( <i>WWDG_HandleTypeDef</i> * hwdg)</code>
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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 45.2.13 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>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 45.2.14 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 using __HAL_WWDG_ENABLE_IT() macro. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.</li> </ul>

#### 45.2.15 HAL\_WWDG\_WakeupCallback

Function Name	<code>void HAL_WWDG_WakeupCallback ( <i>WWDG_HandleTypeDef</i> * hwdg)</code>
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hwdg</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>• None.</li></ul>

#### 45.2.16 HAL\_WWDG\_GetState

Function Name	<code>HAL_WWDG_StateTypeDef HAL_WWDG_GetState ( <i>WWDG_HandleTypeDef</i> * hwdg)</code>
Function Description	Returns the WWDG state.
Parameters	<ul style="list-style-type: none"><li>• <b>hwdg</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>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 45.3 WWDG Firmware driver defines

#### 45.3.1 WWDG

WWDG

**WWDG BitAddress AliasRegion**

CFR\_BASE

**WWDG Counter**

**IS\_WWDG\_COUNTER****WWDG Exported Macros****\_HAL\_WWDG\_RESET\_HANDLE\_STATE****Description:**

- Reset WWDG handle state.

**Parameters:**

- `_HANDLE_`: WWDG handle

**Return value:**

- None:

**\_HAL\_WWDG\_ENABLE****Description:**

- Enables the WWDG peripheral.

**Parameters:**

- `_HANDLE_`: WWDG handle

**Return value:**

- None:

**\_HAL\_WWDG\_GET\_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\_ENABLE\_IT****Description:**

- Enables the WWDG early wakeup interrupt.

**Parameters:**

- \_\_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\_CLEAR\_IT

**Description:**

- Clear the WWDG's interrupt pending bits to clear the selected interrupt pending bits.

**Parameters:**

- \_\_HANDLE\_\_: WWDG handle
- \_\_INTERRUPT\_\_: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - WWDG\_FLAG\_EWIF: Early wakeup interrupt flag

***WWDG Flag definition***

WWDG\_FLAG\_EWIF Early wakeup interrupt flag

***WWDG Interrupt definition***

WWDG\_IT\_EWI

***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

IS\_WWDG\_PRESCALER

***WWDG Window***

IS\_WWDG\_WINDOW

## 46

## FAQs

### General subjects

#### Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not require in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, ...). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL\_UART\_Init() then HAL\_UART\_Transmit() or HAL\_UART\_Receive().

#### Which STM32F0 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32F0 devices. To ensure compatibility between all devices and portability with other 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: `stm32f0xx_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 (stm32f0xx\_hal\_conf\_template.c).

### Which header files should I include in my application to use the HAL drivers?

Only stm32f0xx\_hal.h file has to be included.

### What is the difference between stm32f0xx\_hal\_ppp.c/h and stm32f0xx\_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 (stm32f0xx\_hal\_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32f0xx\_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\_stm32f0xx.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\_MspDelinit() functions?

These function are called within HAL\_PPP\_Init() and HAL\_PPP\_Delinit(), 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 stm32f0xx\_hal\_msp.c. A template is provided in the HAL driver folders (stm32f0xx\_hal\_msp\_template.c).

#### When and how should I use callbacks functions (functions declared with the attribute \_\_weak)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

### **Is it mandatory to use HAL\_Init() function at the beginning of the user application?**

It is mandatory to use HAL\_Init() function to enable the system configuration (Prefetch, Data instruction cache,...), configure the systTick and the NVIC priority grouping and the hardware low level initialization.

The sysTick configuration shall be adjusted by calling **HAL\_RCC\_ClockConfig()** function, to obtain 1 ms whatever the system clock.

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

The SysTick timer is configured to be used to generate variable increments by calling **HAL\_IncTick()** function in Systick ISR and retrieve the value of this variable by calling **HAL\_GetTick()** function.

The call **HAL\_GetTick()** function is mandatory when using HAL drivers with Polling Process or when using **HAL\_Delay()**.

### **Why is the SysTick timer configured to have 1 ms?**

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

### **Could HAL\_Delay() function block my application under certain conditions?**

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

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

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

1. Call **HAL\_Init()** function to initialize the system (data cache, NVIC priority,...).
2. Initialize the system clock by calling **HAL\_RCC\_OscConfig()** followed by **HAL\_RCC\_ClockConfig()**.
3. Add **HAL\_IncTick()** function under **SysTick\_Handler()** ISR function to enable polling process when using **HAL\_Delay()** function
4. Start initializing your peripheral by calling **HAL\_PPP\_Init()**.
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling **HAL\_PPP\_MspInit()** in **stm32f0xx\_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 **stm32f0xx\_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 stm32f0xx\_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.

## 47 Revision history

**Table 25: Document revision history**

Date	Revision	Changes
21-Nov-2014	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.

© 2014 STMicroelectronics – All rights reserved