

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

1	Acronyms and definitions.....	34
2	Overview of HAL drivers	36
2.1	HAL and user-application files.....	36
2.1.1	HAL driver files	36
2.1.2	User-application files	37
2.2	HAL data structures	39
2.2.1	Peripheral handle structures	39
2.2.2	Initialization and configuration structure	40
2.2.3	Specific process structures	41
2.3	API classification	41
2.4	Devices supported by HAL drivers	42
2.5	HAL drivers rules.....	44
2.5.1	HAL API naming rules	44
2.5.2	HAL general naming rules.....	45
2.5.3	HAL interrupt handler and callback functions.....	46
2.6	HAL generic APIs.....	47
2.7	HAL extension APIs	48
2.7.1	HAL extension model overview	48
2.7.2	HAL extension model cases	49
2.8	File inclusion model.....	51
2.9	HAL common resources.....	52
2.10	HAL configuration.....	52
2.11	HAL system peripheral handling	54
2.11.1	Clock.....	54
2.11.2	GPIOs.....	54
2.11.3	Cortex NVIC and SysTick timer.....	56
2.11.4	PWR	56
2.11.5	EXTI.....	57
2.11.6	DMA.....	58
2.12	How to use HAL drivers	59
2.12.1	HAL usage models	59
2.12.2	HAL initialization	60
2.12.3	HAL IO operation process	62
2.12.4	Timeout and error management.....	65
3	HAL System Driver	69

3.1	HAL Firmware driver API description	69
3.1.1	How to use this driver	69
3.1.2	Initialization and de-initialization functions	69
3.1.3	HAL Control functions.....	69
3.1.4	HAL_Init.....	70
3.1.5	HAL_DelInit	70
3.1.6	HAL_MspInit.....	70
3.1.7	HAL_MspDeInit	70
3.1.8	HAL_InitTick	71
3.1.9	HAL_IncTick	71
3.1.10	HAL_GetTick	71
3.1.11	HAL_Delay	71
3.1.12	HAL_SuspendTick.....	72
3.1.13	HAL_ResumeTick.....	72
3.1.14	HAL_GetHalVersion	72
3.1.15	HAL_GetREVID	72
3.1.16	HAL_GetDEVID.....	72
3.1.17	HAL_DBGMCU_EnableDBGStopMode	73
3.1.18	HAL_DBGMCU_DisableDBGStopMode	73
3.1.19	HAL_DBGMCU_EnableDBGStandbyMode	73
3.1.20	HAL_DBGMCU_DisableDBGStandbyMode	73
3.2	HAL Firmware driver defines.....	73
3.2.1	HAL.....	73
4	HAL ADC Generic Driver.....	79
4.1	ADC Firmware driver registers structures	79
4.1.1	ADC_InitTypeDef.....	79
4.1.2	ADC_ChannelConfTypeDef	81
4.1.3	ADC_AnalogWDGConfTypeDef.....	81
4.1.4	ADC_HandleTypeDef	82
4.2	ADC Firmware driver API description.....	83
4.2.1	ADC peripheral features	83
4.2.2	How to use this driver	83
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	87
4.2.10	HAL_ADC_MspDeInit.....	88
4.2.11	HAL_ADC_Start	88
4.2.12	HAL_ADC_Stop.....	88
4.2.13	HAL_ADC_PollForConversion	88
4.2.14	HAL_ADC_PollForEvent	89
4.2.15	HAL_ADC_Start_IT	89
4.2.16	HAL_ADC_Stop_IT	89
4.2.17	HAL_ADC_Start_DMA	89
4.2.18	HAL_ADC_Stop_DMA.....	89
4.2.19	HAL_ADC_GetValue	90
4.2.20	HAL_ADC_IRQHandler.....	90
4.2.21	HAL_ADC_ConvCpltCallback	90
4.2.22	HAL_ADC_ConvHalfCpltCallback.....	90
4.2.23	HAL_ADC_LevelOutOfWindowCallback	90
4.2.24	HAL_ADC_ErrorCallback	90
4.2.25	HAL_ADC_ConfigChannel	91
4.2.26	HAL_ADC_AnalogWDGConfig	91
4.2.27	HAL_ADC_GetState.....	91
4.2.28	HAL_ADC_GetError	92
4.3	ADC Firmware driver defines	92
4.3.1	ADC	92
5	HAL ADC Extension Driver	100
5.1	ADCEx Firmware driver API description	100
5.1.1	IO operation functions	100
5.1.2	HAL_ADCEx_Calibration_Start.....	100
6	HAL CAN Generic Driver	101
6.1	CAN Firmware driver registers structures	101
6.1.1	CAN_InitTypeDef.....	101
6.1.2	CAN_FilterConfTypeDef.....	102
6.1.3	CanTxMsgTypeDef.....	103
6.1.4	CanRxMsgTypeDef	103
6.1.5	CAN_HandleTypeDef	104
6.2	CAN Firmware driver API description.....	105
6.2.1	How to use this driver	105
6.2.2	Initialization and de-initialization functions	106
6.2.3	Peripheral State and Error functions	106
6.2.4	HAL_CAN_Init	106

6.2.5	HAL_CAN_ConfigFilter.....	106
6.2.6	HAL_CAN_DelInit.....	107
6.2.7	HAL_CAN_MspInit	107
6.2.8	HAL_CAN_MspDeInit.....	107
6.2.9	HAL_CAN_Transmit.....	107
6.2.10	HAL_CAN_Transmit_IT.....	107
6.2.11	HAL_CAN_Receive	108
6.2.12	HAL_CAN_Receive_IT.....	108
6.2.13	HAL_CAN_Sleep.....	108
6.2.14	HAL_CAN_WakeUp	108
6.2.15	HAL_CAN_IRQHandler.....	108
6.2.16	HAL_CAN_TxCpltCallback.....	109
6.2.17	HAL_CAN_RxCpltCallback	109
6.2.18	HAL_CAN_ErrorCallback	109
6.2.19	HAL_CAN_GetState.....	109
6.2.20	HAL_CAN_GetError	109
6.3	CAN Firmware driver defines	110
6.3.1	CAN	110
7	HAL CEC Generic Driver	117
7.1	CEC Firmware driver registers structures	117
7.1.1	CEC_InitTypeDef.....	117
7.1.2	CEC_HandleTypeDef	118
7.2	CEC Firmware driver API description.....	119
7.2.1	How to use this driver	119
7.2.2	Initialization and Configuration functions.....	119
7.2.3	IO operation function	119
7.2.4	Peripheral Control function	120
7.2.5	HAL_CEC_Init	120
7.2.6	HAL_CEC_DelInit.....	120
7.2.7	HAL_CEC_MspInit	120
7.2.8	HAL_CEC_MspDeInit.....	120
7.2.9	HAL_CEC_Transmit.....	120
7.2.10	HAL_CEC_Receive	121
7.2.11	HAL_CEC_Transmit_IT.....	121
7.2.12	HAL_CEC_Receive_IT.....	121
7.2.13	HAL_CEC_GetReceivedFrameSize.....	122
7.2.14	HAL_CEC_IRQHandler.....	122
7.2.15	HAL_CEC_TxCpltCallback.....	122

7.2.16	HAL_CEC_RxCpltCallback	122
7.2.17	HAL_CEC_ErrorCallback	122
7.2.18	HAL_CEC_GetState.....	123
7.2.19	HAL_CEC_GetError	123
7.3	CEC Firmware driver defines	123
7.3.1	CEC	123
8	HAL COMP Generic Driver.....	132
8.1	COMP Firmware driver registers structures	132
8.1.1	COMP_InitTypeDef	132
8.1.2	COMP_HandleTypeDef.....	132
8.2	COMP Firmware driver API description	133
8.2.1	COMP Peripheral features	133
8.2.2	How to use this driver	134
8.2.3	Initialization and Configuration functions.....	135
8.2.4	IO operation functions	135
8.2.5	Peripheral Control functions	135
8.2.6	Peripheral State functions	135
8.2.7	HAL_COMP_Init	135
8.2.8	HAL_COMP_DeInit	135
8.2.9	HAL_COMP_MspInit	136
8.2.10	HAL_COMP_MspDeInit.....	136
8.2.11	HAL_COMP_Start	136
8.2.12	HAL_COMP_Stop	136
8.2.13	HAL_COMP_Start_IT	136
8.2.14	HAL_COMP_Stop_IT	137
8.2.15	HAL_COMP_IRQHandler.....	137
8.2.16	HAL_COMP_Lock	137
8.2.17	HAL_COMP_GetOutputLevel	137
8.2.18	HAL_COMP_TriggerCallback	137
8.2.19	HAL_COMP_GetState.....	137
8.3	COMP Firmware driver defines	138
8.3.1	COMP	138
9	HAL CORTEX Generic Driver.....	146
9.1	CORTEX Firmware driver API description	146
9.1.1	How to use this driver	146
9.1.2	Initialization and de-initialization functions	146
9.1.3	Peripheral Control functions	147
9.1.4	HAL_NVIC_SetPriority	147

9.1.5	HAL_NVIC_EnableIRQ	147
9.1.6	HAL_NVIC_DisableIRQ.....	147
9.1.7	HAL_NVIC_SystemReset.....	148
9.1.8	HAL_SYSTICK_Config.....	148
9.1.9	HAL_NVIC_GetPriority.....	148
9.1.10	HAL_NVIC_SetPendingIRQ.....	148
9.1.11	HAL_NVIC_GetPendingIRQ	148
9.1.12	HAL_NVIC_ClearPendingIRQ.....	149
9.1.13	HAL_SYSTICK_CLKSourceConfig	149
9.1.14	HAL_SYSTICK_IRQHandler	149
9.1.15	HAL_SYSTICK_Callback	149
9.2	CORTEX Firmware driver defines.....	149
9.2.1	CORTEX.....	149
10	HAL CRC Generic Driver.....	151
10.1	CRC Firmware driver registers structures	151
10.1.1	CRC_InitTypeDef	151
10.1.2	CRC_HandleTypeDef.....	152
10.2	CRC Firmware driver API description	152
10.2.1	How to use this driver	152
10.2.2	Initialization and Configuration functions.....	153
10.2.3	Peripheral Control functions	153
10.2.4	Peripheral State functions	153
10.2.5	HAL_CRC_Init	153
10.2.6	HAL_CRC_DeInit	153
10.2.7	HAL_CRC_MspInit	154
10.2.8	HAL_CRC_MspDeInit.....	154
10.2.9	HAL_CRC_Accumulate	154
10.2.10	HAL_CRC_Calculate.....	154
10.2.11	HAL_CRC_GetState.....	154
10.3	CRC Firmware driver defines	155
10.3.1	CRC	155
11	HAL CRC Extension Driver	157
11.1	CRCEEx Firmware driver API description	157
11.1.1	How to use this driver	157
11.1.2	Initialization and Configuration functions.....	157
11.1.3	HAL_CRCEEx_Init.....	157
11.1.4	HAL_CRCEEx_Input_Data_Reverse	157

11.1.5	HAL_CRCEx_Output_Data_Reverse.....	158
11.1.6	HAL_CRCEx_Polynomial_Set	158
11.2	CRCEEx Firmware driver defines.....	159
11.2.1	CRCEEx.....	159
12	HAL DAC Generic Driver.....	161
12.1	DAC Firmware driver registers structures	161
12.1.1	DAC_HandleTypeDef	161
12.1.2	DAC_ChannelConfTypeDef	161
12.2	DAC Firmware driver API description.....	162
12.2.1	DAC Peripheral features.....	162
12.2.2	How to use this driver	163
12.2.3	Initialization and de-initialization functions	164
12.2.4	IO operation functions	164
12.2.5	Peripheral Control functions	165
12.2.6	Peripheral State and Errors functions	165
12.2.7	HAL_DAC_Init	165
12.2.8	HAL_DAC_DelInit.....	165
12.2.9	HAL_DAC_MspInit	165
12.2.10	HAL_DAC_MspDelInit.....	166
12.2.11	HAL_DAC_Start	166
12.2.12	HAL_DAC_Stop.....	166
12.2.13	HAL_DAC_Start_DMA	166
12.2.14	HAL_DAC_Stop_DMA.....	167
12.2.15	HAL_DAC_IRQHandler.....	167
12.2.16	HAL_DAC_SetValue	167
12.2.17	HAL_DAC_ConvCpltCallbackCh1.....	168
12.2.18	HAL_DAC_ConvHalfCpltCallbackCh1	168
12.2.19	HAL_DAC_ErrorCallbackCh1	168
12.2.20	HAL_DAC_DMAUnderrunCallbackCh1	168
12.2.21	HAL_DAC_GetValue	168
12.2.22	HAL_DAC_ConfigChannel	169
12.2.23	HAL_DAC_GetState.....	169
12.2.24	HAL_DAC_GetError	169
12.3	DAC Firmware driver defines	169
12.3.1	DAC	169
13	HAL DAC Extension Driver	173
13.1	DACEEx Firmware driver API description	173
13.1.1	How to use this driver	173

13.1.2	Extended features functions	173
13.1.3	HAL_DACEx_DualGetValue	173
13.1.4	HAL_DACEx_TriangleWaveGenerate	173
13.1.5	HAL_DACEx_NoiseWaveGenerate	174
13.1.6	HAL_DACEx_DualSetValue.....	175
13.1.7	HAL_DACEx_ConvCpltCallbackCh2	175
13.1.8	HAL_DACEx_ConvHalfCpltCallbackCh2	175
13.1.9	HAL_DACEx_ErrorCallbackCh2	176
13.1.10	HAL_DACEx_DMAUnderrunCallbackCh2	176
13.1.11	HAL_DACEx_DualGetValue	176
13.2	DACEx Firmware driver defines	176
13.2.1	DACEx.....	176
14	HAL DMA Generic Driver	178
14.1	DMA Firmware driver registers structures	178
14.1.1	DMA_InitTypeDef	178
14.1.2	__DMA_HandleTypeDef.....	178
14.2	DMA Firmware driver API description	179
14.2.1	How to use this driver	179
14.2.2	Initialization and de-initialization functions	180
14.2.3	IO operation functions	180
14.2.4	State and Errors functions	181
14.2.5	HAL_DMA_Init.....	181
14.2.6	HAL_DMA_DeInit	181
14.2.7	HAL_DMA_Start	181
14.2.8	HAL_DMA_Start_IT	182
14.2.9	HAL_DMA_Abort	182
14.2.10	HAL_DMA_PollForTransfer.....	182
14.2.11	HAL_DMA_IRQHandler.....	183
14.2.12	HAL_DMA_GetState	183
14.2.13	HAL_DMA_GetError.....	183
14.3	DMA Firmware driver defines.....	183
14.3.1	DMA.....	183
15	HAL DMA Extension Driver.....	188
15.1	DMAEx Firmware driver defines.....	188
15.1.1	DMAEx.....	188
16	HAL FLASH Generic Driver.....	195
16.1	FLASH Firmware driver registers structures	195

16.1.1	FLASH_ProcessTypeDef	195
16.2	FLASH Firmware driver API description.....	195
16.2.1	FLASH peripheral features	195
16.2.2	How to use this driver.....	196
16.2.3	Peripheral Control functions	196
16.2.4	Peripheral State functions	196
16.2.5	HAL_FLASH_Program	196
16.2.6	HAL_FLASH_Program_IT	197
16.2.7	HAL_FLASH_IRQHandler	197
16.2.8	HAL_FLASH_EndOfOperationCallback	197
16.2.9	HAL_FLASH_OperationErrorHandler	198
16.2.10	HAL_FLASH_Unlock.....	198
16.2.11	HAL_FLASH_Lock	198
16.2.12	HAL_FLASH_OB_Unlock.....	198
16.2.13	HAL_FLASH_OB_Lock	198
16.2.14	HAL_FLASH_OB_Launch.....	198
16.2.15	HAL_FLASH_GetError	199
16.3	FLASH Firmware driver defines	199
16.3.1	FLASH	199
17	HAL FLASH Extension Driver	202
17.1	FLASHEx Firmware driver registers structures	202
17.1.1	FLASH_EraseInitTypeDef	202
17.1.2	FLASH_OBProgramInitTypeDef	202
17.2	FLASHEx Firmware driver API description.....	203
17.2.1	IO operation functions	203
17.2.2	Peripheral Control functions	203
17.2.3	HAL_FLASHEx_Erase	203
17.2.4	HAL_FLASHEx_Erase_IT	204
17.2.5	HAL_FLASHEx_OBErase	204
17.2.6	HAL_FLASHEx_OBProgram.....	204
17.2.7	HAL_FLASHEx_OBGetConfig	204
17.3	FLASHEx Firmware driver defines	205
17.3.1	FLASHEx	205
18	HAL GPIO Generic Driver.....	208
18.1	GPIO Firmware driver registers structures	208
18.1.1	GPIO_InitTypeDef	208
18.2	GPIO Firmware driver API description	208

18.2.1	GPIO Peripheral features	208
18.2.2	How to use this driver	209
18.2.3	Initialization and de-initialization functions	209
18.2.4	IO operation functions	210
18.2.5	HAL_GPIO_Init.....	210
18.2.6	HAL_GPIO_DelInit	210
18.2.7	HAL_GPIO_ReadPin.....	210
18.2.8	HAL_GPIO_WritePin	211
18.2.9	HAL_GPIO_TogglePin	211
18.2.10	HAL_GPIO_LockPin.....	211
18.2.11	HAL_GPIO_EXTI_IRQHandler	211
18.2.12	HAL_GPIO_EXTI_Callback.....	212
18.3	GPIO Firmware driver defines.....	212
18.3.1	GPIO.....	212
19	HAL GPIO Extension Driver.....	215
19.1	GPIOEx Firmware driver defines.....	215
19.1.1	GPIOEx	215
20	HAL I2C Generic Driver.....	218
20.1	I2C Firmware driver registers structures	218
20.1.1	I2C_InitTypeDef.....	218
20.1.2	I2C_HandleTypeDef	218
20.2	I2C Firmware driver API description.....	219
20.2.1	How to use this driver	219
20.2.2	Initialization and de-initialization functions	222
20.2.3	IO operation functions	223
20.2.4	Peripheral State and Errors functions	224
20.2.5	HAL_I2C_Init	224
20.2.6	HAL_I2C_DelInit.....	224
20.2.7	HAL_I2C_MspInit	224
20.2.8	HAL_I2C_MspDeInit.....	224
20.2.9	HAL_I2C_Master_Transmit.....	225
20.2.10	HAL_I2C_Master_Receive	225
20.2.11	HAL_I2C_Slave_Transmit.....	225
20.2.12	HAL_I2C_Slave_Receive	225
20.2.13	HAL_I2C_Master_Transmit_IT.....	226
20.2.14	HAL_I2C_Master_Receive_IT.....	226
20.2.15	HAL_I2C_Slave_Transmit_IT.....	226

20.2.16	HAL_I2C_Slave_Receive_IT.....	227
20.2.17	HAL_I2C_Master_Transmit_DMA.....	227
20.2.18	HAL_I2C_Master_Receive_DMA.....	227
20.2.19	HAL_I2C_Slave_Transmit_DMA.....	227
20.2.20	HAL_I2C_Slave_Receive_DMA.....	228
20.2.21	HAL_I2C_Mem_Write.....	228
20.2.22	HAL_I2C_Mem_Read	228
20.2.23	HAL_I2C_Mem_Write_IT	229
20.2.24	HAL_I2C_Mem_Read_IT	229
20.2.25	HAL_I2C_Mem_Write_DMA	229
20.2.26	HAL_I2C_Mem_Read_DMA	230
20.2.27	HAL_I2C_IsDeviceReady.....	230
20.2.28	HAL_I2C_EV_IRQHandler	230
20.2.29	HAL_I2C_ER_IRQHandler	231
20.2.30	HAL_I2C_MasterTxCpltCallback.....	231
20.2.31	HAL_I2C_MasterRxCpltCallback	231
20.2.32	HAL_I2C_SlaveTxCpltCallback.....	231
20.2.33	HAL_I2C_SlaveRxCpltCallback	232
20.2.34	HAL_I2C_MemTxCpltCallback.....	232
20.2.35	HAL_I2C_MemRxCpltCallback	232
20.2.36	HAL_I2C_ErrorCallback	232
20.2.37	HAL_I2C_GetState	232
20.2.38	HAL_I2C_GetError	233
20.3	I2C Firmware driver defines	233
20.3.1	I2C	233
21	HAL I2C Extension Driver	239
21.1	I2CEx Firmware driver API description	239
21.1.1	I2C peripheral Extended features.....	239
21.1.2	How to use this driver	239
21.1.3	Extended features functions	239
21.1.4	HAL_I2CEEx_ConfigAnalogFilter	239
21.1.5	HAL_I2CEEx_ConfigDigitalFilter	240
21.1.6	HAL_I2CEEx_EnableWakeUp.....	240
21.1.7	HAL_I2CEEx_DisableWakeUp	240
21.1.8	HAL_I2CEEx_EnableFastModePlus	240
21.1.9	HAL_I2CEEx_DisableFastModePlus.....	241
21.2	I2CEx Firmware driver defines	241
21.2.1	I2CEx	241

22 HAL I2S Generic Driver	242
22.1 I2S Firmware driver registers structures	242
22.1.1 I2S_InitTypeDef.....	242
22.1.2 I2S_HandleTypeDef	242
22.2 I2S Firmware driver API description.....	243
22.2.1 How to use this driver	243
22.2.2 Initialization and de-initialization functions	245
22.2.3 IO operation functions	246
22.2.4 Peripheral State and Errors functions	246
22.2.5 HAL_I2S_Init	246
22.2.6 HAL_I2S_DeInit.....	247
22.2.7 HAL_I2S_MspInit.....	247
22.2.8 HAL_I2S_MspDeInit.....	247
22.2.9 HAL_I2S_Transmit	247
22.2.10 HAL_I2S_Receive	248
22.2.11 HAL_I2S_Transmit_IT	248
22.2.12 HAL_I2S_Receive_IT	249
22.2.13 HAL_I2S_Transmit_DMA	249
22.2.14 HAL_I2S_Receive_DMA	249
22.2.15 HAL_I2S_DMAPause	250
22.2.16 HAL_I2S_DMAResume.....	250
22.2.17 HAL_I2S_DMAStop.....	250
22.2.18 HAL_I2S_IRQHandler	250
22.2.19 HAL_I2S_TxHalfCpltCallback	251
22.2.20 HAL_I2S_TxCpltCallback	251
22.2.21 HAL_I2S_RxHalfCpltCallback	251
22.2.22 HAL_I2S_RxCpltCallback	251
22.2.23 HAL_I2S_ErrorCallback	251
22.2.24 HAL_I2S_GetState	251
22.2.25 HAL_I2S_GetError	252
22.3 I2S Firmware driver defines	252
22.3.1 I2S	252
23 HAL IRDA Generic Driver	256
23.1 IRDA Firmware driver registers structures	256
23.1.1 IRDA_InitTypeDef.....	256
23.1.2 IRDA_HandleTypeDef	256
23.2 IRDA Firmware driver API description.....	257

23.2.1	How to use this driver	257
23.2.2	Initialization and Configuration functions	259
23.2.3	IO operation functions	260
23.2.4	Peripheral State and Errors functions	261
23.2.5	HAL_IRDA_Init	261
23.2.6	HAL_IRDA_DelInit.....	261
23.2.7	HAL_IRDA_MspInit	262
23.2.8	HAL_IRDA_MspDelInit.....	262
23.2.9	HAL_IRDA_Transmit.....	262
23.2.10	HAL_IRDA_Receive	262
23.2.11	HAL_IRDA_Transmit_IT	263
23.2.12	HAL_IRDA_Receive_IT.....	263
23.2.13	HAL_IRDA_Transmit_DMA	263
23.2.14	HAL_IRDA_Receive_DMA.....	263
23.2.15	HAL_IRDA_DMAPause.....	264
23.2.16	HAL_IRDA_DMAResume.....	264
23.2.17	HAL_IRDA_DMAStop.....	264
23.2.18	HAL_IRDA_IRQHandler	264
23.2.19	HAL_IRDA_TxCpltCallback.....	265
23.2.20	HAL_IRDA_TxHalfCpltCallback	265
23.2.21	HAL_IRDA_RxCpltCallback	265
23.2.22	HAL_IRDA_RxHalfCpltCallback.....	265
23.2.23	HAL_IRDA_ErrorCallback	265
23.2.24	HAL_IRDA_GetState.....	266
23.2.25	HAL_IRDA_GetError	266
23.3	IRDA Firmware driver defines	266
23.3.1	IRDA	266
24	HAL IRDA Extension Driver	275
24.1	IRDAEx Firmware driver defines	275
24.1.1	IRDAEx.....	275
25	HAL IWDG Generic Driver.....	276
25.1	IWDG Firmware driver registers structures	276
25.1.1	IWDG_InitTypeDef	276
25.1.2	IWDG_HandleTypeDef.....	276
25.2	IWDG Firmware driver API description	277
25.2.1	IWDG Specific features	277
25.2.2	How to use this driver	277
25.2.3	Initialization and de-initialization functions	278

25.2.4	IO operation functions	278
25.2.5	Peripheral State functions	278
25.2.6	HAL_IWDG_Init.....	278
25.2.7	HAL_IWDG_MspInit	279
25.2.8	HAL_IWDG_Start	279
25.2.9	HAL_IWDG_Refresh	279
25.2.10	HAL_IWDG_GetState.....	279
25.3	IWDG Firmware driver defines	279
25.3.1	IWDG	279
26	HAL PCD Generic Driver	282
26.1	PCD Firmware driver registers structures	282
26.1.1	PCD_InitTypeDef.....	282
26.1.2	PCD_EPTTypeDef.....	282
26.1.3	PCD_HandleTypeDef.....	283
26.2	PCD Firmware driver API description.....	284
26.2.1	How to use this driver	284
26.2.2	Initialization and de-initialization functions	285
26.2.3	IO operation functions	285
26.2.4	Peripheral Control functions	285
26.2.5	Peripheral State functions	285
26.2.6	HAL_PCD_Init	286
26.2.7	HAL_PCD_DelInit.....	286
26.2.8	HAL_PCD_MspInit	286
26.2.9	HAL_PCD_MspDeInit.....	286
26.2.10	HAL_PCD_Start	286
26.2.11	HAL_PCD_Stop.....	286
26.2.12	HAL_PCD_IRQHandler.....	287
26.2.13	HAL_PCD_DataOutStageCallback	287
26.2.14	HAL_PCD_DataInStageCallback	287
26.2.15	HAL_PCD_SetupStageCallback	287
26.2.16	HAL_PCD_SOFCallback.....	287
26.2.17	HAL_PCD_ResetCallback.....	287
26.2.18	HAL_PCD_SuspendCallback.....	288
26.2.19	HAL_PCD_ResumeCallback.....	288
26.2.20	HAL_PCD_ISOOUTIncompleteCallback.....	288
26.2.21	HAL_PCD_ISOINIncompleteCallback.....	288
26.2.22	HAL_PCD_ConnectCallback.....	288
26.2.23	HAL_PCD_DisconnectCallback	289

26.2.24	HAL_PCD_DevConnect	289
26.2.25	HAL_PCD_DevDisconnect.....	289
26.2.26	HAL_PCD_SetAddress	289
26.2.27	HAL_PCD_EP_Open	289
26.2.28	HAL_PCD_EP_Close	289
26.2.29	HAL_PCD_EP_Receive	290
26.2.30	HAL_PCD_EP_GetRxCount	290
26.2.31	HAL_PCD_EP_Transmit	290
26.2.32	HAL_PCD_EP_SetStall.....	290
26.2.33	HAL_PCD_EP_ClrStall.....	291
26.2.34	HAL_PCD_EP_Flush	291
26.2.35	HAL_PCD_ActivateRemoteWakeUp	291
26.2.36	HAL_PCD_DeActivateRemoteWakeUp.....	291
26.2.37	HAL_PCD_GetState.....	291
26.3	PCD Firmware driver defines	292
26.3.1	PCD	292
27	HAL PCD Extension Driver	294
27.1	PCDEx Firmware driver API description	294
27.1.1	Extended Peripheral Control functions.....	294
27.1.2	HAL_PCDEx_PMAConfig	294
28	HAL PWR Generic Driver	295
28.1	PWR Firmware driver API description.....	295
28.1.1	Initialization and de-initialization functions	295
28.1.2	Peripheral Control functions	295
28.1.3	HAL_PWR_EnableBkUpAccess	297
28.1.4	HAL_PWR_DisableBkUpAccess.....	297
28.1.5	HAL_PWR_EnableWakeUpPin.....	297
28.1.6	HAL_PWR_DisableWakeUpPin.....	297
28.1.7	HAL_PWR_EnterSLEEPMode.....	298
28.1.8	HAL_PWR_EnterSTOPMode.....	298
28.1.9	HAL_PWR_EnterSTANDBYMode	299
28.1.10	HAL_PWR_EnableSleepOnExit.....	299
28.1.11	HAL_PWR_DisableSleepOnExit	299
28.1.12	HAL_PWR_EnableSEVOnPend	299
28.1.13	HAL_PWR_DisableSEVOnPend.....	300
28.1.14	HAL_PWR_EnableBkUpAccess	300
28.1.15	HAL_PWR_DisableBkUpAccess.....	300
28.2	PWR Firmware driver defines	300

28.2.1	PWR	300
29	HAL PWR Extension Driver	302
29.1	PWREEx Firmware driver registers structures	302
29.1.1	PWR_PVDTTypeDef	302
29.2	PWREEx Firmware driver API description.....	302
29.2.1	Peripheral extended control functions	302
29.2.2	HAL_PWR_ConfigPVD	303
29.2.3	HAL_PWR_EnablePVD.....	303
29.2.4	HAL_PWR_DisablePVD.....	303
29.2.5	HAL_PWR_PVD_IRQHandler.....	303
29.2.6	HAL_PWR_PVDCALLBACK	303
29.2.7	HAL_PWREEx_EnableVddio2Monitor	303
29.2.8	HAL_PWREEx_DisableVddio2Monitor.....	304
29.2.9	HAL_PWREEx_Vddio2Monitor_IRQHandler.....	304
29.2.10	HAL_PWREEx_Vddio2MonitorCallback.....	304
29.3	PWREEx Firmware driver defines	304
29.3.1	PWREEx	304
30	HAL RCC Generic Driver.....	309
30.1	RCC Firmware driver registers structures	309
30.1.1	RCC_PLLInitTypeDef	309
30.1.2	RCC_OscInitTypeDef	309
30.1.3	RCC_ClkInitTypeDef	310
30.2	RCC Firmware driver API description	311
30.2.1	RCC specific features	311
30.2.2	RCC Limitations.....	311
30.2.3	Initialization and de-initialization function	311
30.2.4	Peripheral Control functions	312
30.2.5	HAL_RCC_DeInit	313
30.2.6	HAL_RCC_OscConfig	313
30.2.7	HAL_RCC_ClockConfig	313
30.2.8	HAL_RCC_MCOConfig	314
30.2.9	HAL_RCC_EnableCSS	314
30.2.10	HAL_RCC_DisableCSS	314
30.2.11	HAL_RCC_GetSysClockFreq	315
30.2.12	HAL_RCC_GetHCLKFreq.....	315
30.2.13	HAL_RCC_GetPCLK1Freq	315
30.2.14	HAL_RCC_GetOscConfig	316

30.2.15	HAL_RCC_GetClockConfig	316
30.2.16	HAL_RCC_NMI_IRQHandler	316
30.2.17	HAL_RCC_CSSCallback.....	316
30.3	RCC Firmware driver defines	317
30.3.1	RCC	317
31	HAL RCC Extension Driver	336
31.1	RCCEEx Firmware driver registers structures	336
31.1.1	RCC_PерiphCLKInitTypeDef	336
31.1.2	RCC_CRSInitTypeDef	336
31.1.3	RCC_CRSSynchroInfoTypeDef	337
31.2	RCCEEx Firmware driver API description	338
31.2.1	Extended Peripheral Control functions	338
31.2.2	HAL_RCCEEx_PeriphCLKConfig	338
31.2.3	HAL_RCCEEx_GetPeriphCLKConfig	338
31.2.4	HAL_RCCEEx_GetPeriphCLKFreq	339
31.2.5	HAL_RCCEEx_CRSConfig	339
31.2.6	HAL_RCCEEx_CRSSoftwareSynchronizationGenerate	339
31.2.7	HAL_RCCEEx_CRSGetSynchronizationInfo	339
31.2.8	HAL_RCCEEx_CRSWaitSynchronization	340
31.3	RCCEEx Firmware driver defines	340
31.3.1	RCCEx	340
32	HAL RTC Generic Driver	353
32.1	RTC Firmware driver registers structures	353
32.1.1	RTC_InitTypeDef	353
32.1.2	RTC_TimeTypeDef	353
32.1.3	RTC_DateTypeDef	354
32.1.4	RTC_AlarmTypeDef	355
32.1.5	RTC_HandleTypeDef	355
32.2	RTC Firmware driver API description	356
32.2.1	How to use RTC Driver	356
32.2.2	RTC and low power modes	356
32.2.3	Initialization and de-initialization functions	356
32.2.4	RTC Time and Date functions	357
32.2.5	RTC Alarm functions	357
32.2.6	Peripheral Control functions	357
32.2.7	Peripheral State functions	357
32.2.8	HAL_RTC_Init	358

32.2.9	HAL_RTC_DelInit.....	358
32.2.10	HAL_RTC_MspInit.....	358
32.2.11	HAL_RTC_MspDeInit	358
32.2.12	HAL_RTC_SetTime.....	358
32.2.13	HAL_RTC_GetTime	359
32.2.14	HAL_RTC_SetDate	359
32.2.15	HAL_RTC_GetDate.....	359
32.2.16	HAL_RTC_SetAlarm	360
32.2.17	HAL_RTC_SetAlarm_IT	360
32.2.18	HAL_RTC_DeactivateAlarm.....	360
32.2.19	HAL_RTC_GetAlarm	360
32.2.20	HAL_RTC_AlarmIRQHandler.....	361
32.2.21	HAL_RTC_AlarmAEventCallback	361
32.2.22	HAL_RTC_PollForAlarmAEvent.....	361
32.2.23	HAL_RTC_WaitForSynchro	361
32.2.24	HAL_RTC_GetState	362
32.3	RTC Firmware driver defines	362
32.3.1	RTC	362
33	HAL RTC Extension Driver	372
33.1	RTCEx Firmware driver registers structures	372
33.1.1	RTC_TamperTypeDef	372
33.2	RTCEx Firmware driver API description.....	372
33.2.1	How to use this driver	372
33.2.2	RTC TimeStamp and Tamper functions.....	373
33.2.3	RTC Wake-up functions	374
33.2.4	Extended Peripheral Control functions.....	374
33.2.5	HAL_RTCEx_SetTimeStamp	374
33.2.6	HAL_RTCEx_SetTimeStamp_IT	375
33.2.7	HAL_RTCEx_DeactivateTimeStamp	375
33.2.8	HAL_RTCEx_GetTimeStamp.....	375
33.2.9	HAL_RTCEx_SetTamper	376
33.2.10	HAL_RTCEx_SetTamper_IT	376
33.2.11	HAL_RTCEx_DeactivateTamper	376
33.2.12	HAL_RTCEx_TamperTimeStampIRQHandler	377
33.2.13	HAL_RTCEx_TimeStampEventCallback	377
33.2.14	HAL_RTCEx_Tamper1EventCallback	377
33.2.15	HAL_RTCEx_Tamper2EventCallback	377
33.2.16	HAL_RTCEx_Tamper3EventCallback	377

33.2.17	HAL_RTCEx_PollForTimeStampEvent.....	377
33.2.18	HAL_RTCEx_PollForTamper1Event.....	378
33.2.19	HAL_RTCEx_PollForTamper2Event.....	378
33.2.20	HAL_RTCEx_PollForTamper3Event.....	378
33.2.21	HAL_RTCEx_SetWakeUpTimer	378
33.2.22	HAL_RTCEx_SetWakeUpTimer_IT	378
33.2.23	HAL_RTCEx_DeactivateWakeUpTimer.....	379
33.2.24	HAL_RTCEx_GetWakeUpTimer	379
33.2.25	HAL_RTCEx_WakeUpTimerIRQHandler.....	379
33.2.26	HAL_RTCEx_WakeUpTimerEventCallback.....	379
33.2.27	HAL_RTCEx_PollForWakeUpTimerEvent	379
33.2.28	HAL_RTCEx_BKUPWrite.....	380
33.2.29	HAL_RTCEx_BKUPRead	380
33.2.30	HAL_RTCEx_SetSmoothCalib.....	380
33.2.31	HAL_RTCEx_SetSynchroShift	381
33.2.32	HAL_RTCEx_SetCalibrationOutPut	381
33.2.33	HAL_RTCEx_DeactivateCalibrationOutPut	381
33.2.34	HAL_RTCEx_SetRefClock	382
33.2.35	HAL_RTCEx_DeactivateRefClock	382
33.2.36	HAL_RTCEx_EnableBypassShadow	382
33.2.37	HAL_RTCEx_DisableBypassShadow	382
33.3	RTCEEx Firmware driver defines	383
33.3.1	RTCEEx	383
34	HAL SMARTCARD Generic Driver.....	401
34.1	SMARTCARD Firmware driver registers structures	401
34.1.1	SMARTCARD_InitTypeDef	401
34.1.2	SMARTCARD_AdvFeatureInitTypeDef.....	402
34.1.3	SMARTCARD_HandleTypeDef.....	403
34.2	SMARTCARD Firmware driver API description.....	404
34.2.1	How to use this driver	404
34.2.2	Initialization and Configuration functions	406
34.2.3	IO operation functions	407
34.2.4	Peripheral State and Errors functions	408
34.2.5	HAL_SMARTCARD_Init.....	408
34.2.6	HAL_SMARTCARD_DeInit	408
34.2.7	HAL_SMARTCARD_MspInit	409
34.2.8	HAL_SMARTCARD_MspDeInit	409
34.2.9	HAL_SMARTCARD_Transmit.....	409

34.2.10	HAL_SMARTCARD_Receive.....	409
34.2.11	HAL_SMARTCARD_Transmit_IT	410
34.2.12	HAL_SMARTCARD_Receive_IT	410
34.2.13	HAL_SMARTCARD_Transmit_DMA.....	410
34.2.14	HAL_SMARTCARD_Receive_DMA.....	410
34.2.15	HAL_SMARTCARD_IRQHandler.....	411
34.2.16	HAL_SMARTCARD_TxCpltCallback	411
34.2.17	HAL_SMARTCARD_RxCpltCallback	411
34.2.18	HAL_SMARTCARD_ErrorCallback	411
34.2.19	HAL_SMARTCARD_GetState	412
34.2.20	HAL_SMARTCARD_GetError.....	412
34.3	SMARTCARD Firmware driver defines	412
34.3.1	SMARTCARD.....	412
35	HAL SMARTCARD Extension Driver	424
35.1	SMARTCARDEX Firmware driver API description	424
35.1.1	Peripheral Control functions	424
35.1.2	HAL_SMARTCARDEX_BlockLength_Config	424
35.1.3	HAL_SMARTCARDEX_TimeOut_Config	424
35.1.4	HAL_SMARTCARDEX_EnableReceiverTimeOut	425
35.1.5	HAL_SMARTCARDEX_DisableReceiverTimeOut	425
36	HAL SMBUS Generic Driver	426
36.1	SMBUS Firmware driver registers structures	426
36.1.1	SMBUS_InitTypeDef	426
36.1.2	SMBUS_HandleTypeDef.....	427
36.2	SMBUS Firmware driver API description	428
36.2.1	How to use this driver	428
36.2.2	Initialization and de-initialization functions	429
36.2.3	IO operation functions	430
36.2.4	Peripheral State and Errors functions	431
36.2.5	HAL_SMBUS_Init	431
36.2.6	HAL_SMBUS_DelInit	431
36.2.7	HAL_SMBUS_MspInit	431
36.2.8	HAL_SMBUS_MspDeInit.....	431
36.2.9	HAL_SMBUS_Master_Transmit_IT	432
36.2.10	HAL_SMBUS_Master_Receive_IT	432
36.2.11	HAL_SMBUS_Master_Abort_IT	432
36.2.12	HAL_SMBUS_Slave_Transmit_IT	433

36.2.13	HAL_SMBUS_Slave_Receive_IT	433
36.2.14	HAL_SMBUS_EnableListen_IT.....	433
36.2.15	HAL_SMBUS_DisableListen_IT	433
36.2.16	HAL_SMBUS_EnableAlert_IT	434
36.2.17	HAL_SMBUS_DisableAlert_IT	434
36.2.18	HAL_SMBUS_IsDeviceReady	434
36.2.19	HAL_SMBUS_EV_IRQHandler.....	434
36.2.20	HAL_SMBUS_ER_IRQHandler.....	435
36.2.21	HAL_SMBUS_MasterTxCpltCallback	435
36.2.22	HAL_SMBUS_MasterRxCpltCallback	435
36.2.23	HAL_SMBUS_SlaveTxCpltCallback	435
36.2.24	HAL_SMBUS_SlaveRxCpltCallback	435
36.2.25	HAL_SMBUS_AddrCallback	436
36.2.26	HAL_SMBUS_ListenCpltCallback	436
36.2.27	HAL_SMBUS_ErrorCallback	436
36.2.28	HAL_SMBUS_GetState.....	436
36.2.29	HAL_SMBUS_GetError.....	437
36.3	SMBUS Firmware driver defines	437
36.3.1	SMBUS	437
37	HAL SPI Generic Driver.....	444
37.1	SPI Firmware driver registers structures	444
37.1.1	SPI_InitTypeDef	444
37.1.2	_SPI_HandleTypeDef.....	445
37.2	SPI Firmware driver API description	446
37.2.1	How to use this driver	446
37.2.2	Initialization and de-initialization functions	447
37.2.3	IO operation functions	448
37.2.4	Peripheral State and Errors functions	448
37.2.5	HAL_SPI_Init	449
37.2.6	HAL_SPI_DeInit	449
37.2.7	HAL_SPI_MspInit	449
37.2.8	HAL_SPI_MspDeInit.....	449
37.2.9	HAL_SPI_Transmit.....	449
37.2.10	HAL_SPI_Receive.....	450
37.2.11	HAL_SPI_TransmitReceive.....	450
37.2.12	HAL_SPI_Transmit_IT.....	450
37.2.13	HAL_SPI_Receive_IT.....	450
37.2.14	HAL_SPI_TransmitReceive_IT	451

37.2.15	HAL_SPI_Transmit_DMA.....	451
37.2.16	HAL_SPI_Receive_DMA.....	451
37.2.17	HAL_SPI_TransmitReceive_DMA.....	451
37.2.18	HAL_SPI_DMAPause.....	452
37.2.19	HAL_SPI_DMAResume	452
37.2.20	HAL_SPI_DMADelete	452
37.2.21	HAL_SPI_IRQHandler	452
37.2.22	HAL_SPI_TxCpltCallback	452
37.2.23	HAL_SPI_RxCpltCallback	453
37.2.24	HAL_SPI_TxRxCpltCallback	453
37.2.25	HAL_SPI_TxHalfCpltCallback	453
37.2.26	HAL_SPI_RxHalfCpltCallback.....	453
37.2.27	HAL_SPI_TxRxHalfCpltCallback.....	453
37.2.28	HAL_SPI_ErrorCallback	454
37.2.29	HAL_SPI_GetState.....	454
37.2.30	HAL_SPI_GetError	454
37.3	SPI Firmware driver defines	454
37.3.1	SPI.....	454
38	HAL SPI Extension Driver	461
38.1	SPIEx Firmware driver API description	461
38.1.1	IO operation functions	461
38.1.2	HAL_SPIEx_FlushRxFifo	461
39	HAL TIM Generic Driver	462
39.1	TIM Firmware driver registers structures	462
39.1.1	TIM_Base_InitTypeDef.....	462
39.1.2	TIM_OC_InitTypeDef.....	462
39.1.3	TIM_OnePulse_InitTypeDef	463
39.1.4	TIM_IC_InitTypeDef	464
39.1.5	TIM_Encoder_InitTypeDef	464
39.1.6	TIM_ClockConfigTypeDef	465
39.1.7	TIM_ClearInputConfigTypeDef.....	466
39.1.8	TIM_SlaveConfigTypeDef	466
39.1.9	TIM_HandleTypeDef	467
39.2	TIM Firmware driver API description	467
39.2.1	TIMER Generic features	467
39.2.2	How to use this driver	468
39.2.3	Time Base functions	468

39.2.4	Time Output Compare functions	469
39.2.5	Time PWM functions	469
39.2.6	Time Input Capture functions	470
39.2.7	Time One Pulse functions	470
39.2.8	Time Encoder functions.....	471
39.2.9	IRQ handler management	471
39.2.10	Peripheral Control functions	471
39.2.11	TIM Callbacks functions	472
39.2.12	Peripheral State functions	472
39.2.13	HAL_TIM_Base_Init	472
39.2.14	HAL_TIM_Base_DelInit.....	473
39.2.15	HAL_TIM_Base_MspInit.....	473
39.2.16	HAL_TIM_Base_MspDelInit.....	473
39.2.17	HAL_TIM_Base_Start.....	473
39.2.18	HAL_TIM_Base_Stop.....	473
39.2.19	HAL_TIM_Base_Start_IT	473
39.2.20	HAL_TIM_Base_Stop_IT.....	474
39.2.21	HAL_TIM_Base_Start_DMA	474
39.2.22	HAL_TIM_Base_Stop_DMA.....	474
39.2.23	HAL_TIM_OC_Init	474
39.2.24	HAL_TIM_OC_DelInit.....	474
39.2.25	HAL_TIM_OC_MspInit	475
39.2.26	HAL_TIM_OC_MspDelInit.....	475
39.2.27	HAL_TIM_OC_Start	475
39.2.28	HAL_TIM_OC_Stop.....	475
39.2.29	HAL_TIM_OC_Start_IT	475
39.2.30	HAL_TIM_OC_Stop_IT	476
39.2.31	HAL_TIM_OC_Start_DMA	476
39.2.32	HAL_TIM_OC_Stop_DMA	476
39.2.33	HAL_TIM_PWM_Init.....	477
39.2.34	HAL_TIM_PWM_DelInit.....	477
39.2.35	HAL_TIM_PWM_MspInit.....	477
39.2.36	HAL_TIM_PWM_MspDelInit	477
39.2.37	HAL_TIM_PWM_Start.....	477
39.2.38	HAL_TIM_PWM_Stop	478
39.2.39	HAL_TIM_PWM_Start_IT	478
39.2.40	HAL_TIM_PWM_Stop_IT	478
39.2.41	HAL_TIM_PWM_Start_DMA.....	478
39.2.42	HAL_TIM_PWM_Stop_DMA.....	479

39.2.43	HAL_TIM_IC_Init	479
39.2.44	HAL_TIM_IC_DelInit	479
39.2.45	HAL_TIM_IC_MspInit	479
39.2.46	HAL_TIM_IC_MspDeInit	480
39.2.47	HAL_TIM_IC_Start	480
39.2.48	HAL_TIM_IC_Stop	480
39.2.49	HAL_TIM_IC_Start_IT	480
39.2.50	HAL_TIM_IC_Stop_IT	481
39.2.51	HAL_TIM_IC_Start_DMA	481
39.2.52	HAL_TIM_IC_Stop_DMA	481
39.2.53	HAL_TIM_OnePulse_Init	481
39.2.54	HAL_TIM_OnePulse_DelInit	482
39.2.55	HAL_TIM_OnePulse_MspInit	482
39.2.56	HAL_TIM_OnePulse_MspDeInit	482
39.2.57	HAL_TIM_OnePulse_Start	482
39.2.58	HAL_TIM_OnePulse_Stop	483
39.2.59	HAL_TIM_OnePulse_Start_IT	483
39.2.60	HAL_TIM_OnePulse_Stop_IT	483
39.2.61	HAL_TIM_Encoder_Init	483
39.2.62	HAL_TIM_Encoder_DelInit	484
39.2.63	HAL_TIM_Encoder_MspInit	484
39.2.64	HAL_TIM_Encoder_MspDeInit	484
39.2.65	HAL_TIM_Encoder_Start	484
39.2.66	HAL_TIM_Encoder_Stop	484
39.2.67	HAL_TIM_Encoder_Start_IT	485
39.2.68	HAL_TIM_Encoder_Stop_IT	485
39.2.69	HAL_TIM_Encoder_Start_DMA	485
39.2.70	HAL_TIM_Encoder_Stop_DMA	486
39.2.71	HAL_TIM_IRQHandler	486
39.2.72	HAL_TIM_OC_ConfigChannel	486
39.2.73	HAL_TIM_IC_ConfigChannel	486
39.2.74	HAL_TIM_PWM_ConfigChannel	487
39.2.75	HAL_TIM_OnePulse_ConfigChannel	487
39.2.76	HAL_TIM_DMABurst_WriteStart	487
39.2.77	HAL_TIM_DMABurst_WriteStop	488
39.2.78	HAL_TIM_DMABurst_ReadStart	488
39.2.79	HAL_TIM_DMABurst_ReadStop	489
39.2.80	HAL_TIM_GenerateEvent	489

39.2.81	HAL_TIM_ConfigOCrefClear.....	489
39.2.82	HAL_TIM_ConfigClockSource	490
39.2.83	HAL_TIM_ConfigTI1Input.....	490
39.2.84	HAL_TIM_SlaveConfigSynchronization	490
39.2.85	HAL_TIM_SlaveConfigSynchronization_IT	491
39.2.86	HAL_TIM_ReadCapturedValue.....	491
39.2.87	HAL_TIM_PeriodElapsedCallback	491
39.2.88	HAL_TIM_OC_DelayElapsedCallback.....	491
39.2.89	HAL_TIM_IC_CaptureCallback	492
39.2.90	HAL_TIM_PWM_PulseFinishedCallback	492
39.2.91	HAL_TIM_TriggerCallback	492
39.2.92	HAL_TIM_ErrorCallback.....	492
39.2.93	HAL_TIM_Base_GetState	492
39.2.94	HAL_TIM_OC_GetState.....	492
39.2.95	HAL_TIM_PWM_GetState	493
39.2.96	HAL_TIM_IC_GetState.....	493
39.2.97	HAL_TIM_OnePulse_GetState	493
39.2.98	HAL_TIM_Encoder_GetState.....	493
39.3	TIM Firmware driver defines.....	493
39.3.1	TIM.....	493
40	HAL TIM Extension Driver.....	511
40.1	TIMEx Firmware driver registers structures.....	511
40.1.1	TIM_HallSensor_InitTypeDef	511
40.1.2	TIM_MasterConfigTypeDef	511
40.1.3	TIM_BreakDeadTimeConfigTypeDef	511
40.2	TIMEx Firmware driver API description	512
40.2.1	TIMER Extended features	512
40.2.2	How to use this driver	512
40.2.3	Timer Hall Sensor functions	513
40.2.4	Timer Complementary Output Compare functions.....	514
40.2.5	Timer Complementary PWM functions.....	514
40.2.6	Timer Complementary One Pulse functions.....	514
40.2.7	Peripheral Control functions	515
40.2.8	Extension Callbacks functions.....	515
40.2.9	Extension Peripheral State functions	515
40.2.10	HAL_TIMEx_HallSensor_Init.....	515
40.2.11	HAL_TIMEx_HallSensor_DelInit	515
40.2.12	HAL_TIMEx_HallSensor_MspInit.....	516

40.2.13	HAL_TIMEx_HallSensor_MspDeInit	516
40.2.14	HAL_TIMEx_HallSensor_Start.....	516
40.2.15	HAL_TIMEx_HallSensor_Stop	516
40.2.16	HAL_TIMEx_HallSensor_Start_IT.....	516
40.2.17	HAL_TIMEx_HallSensor_Stop_IT.....	517
40.2.18	HAL_TIMEx_HallSensor_Start_DMA.....	517
40.2.19	HAL_TIMEx_HallSensor_Stop_DMA.....	517
40.2.20	HAL_TIMEx_OCN_Start.....	517
40.2.21	HAL_TIMEx_OCN_Stop.....	517
40.2.22	HAL_TIMEx_OCN_Start_IT	518
40.2.23	HAL_TIMEx_OCN_Stop_IT	518
40.2.24	HAL_TIMEx_OCN_Start_DMA	518
40.2.25	HAL_TIMEx_OCN_Stop_DMA.....	519
40.2.26	HAL_TIMEx_PWMN_Start	519
40.2.27	HAL_TIMEx_PWMN_Stop	519
40.2.28	HAL_TIMEx_PWMN_Start_IT.....	519
40.2.29	HAL_TIMEx_PWMN_Stop_IT	520
40.2.30	HAL_TIMEx_PWMN_Start_DMA	520
40.2.31	HAL_TIMEx_PWMN_Stop_DMA	520
40.2.32	HAL_TIMEx_OnePulseN_Start	521
40.2.33	HAL_TIMEx_OnePulseN_Stop	521
40.2.34	HAL_TIMEx_OnePulseN_Start_IT	521
40.2.35	HAL_TIMEx_OnePulseN_Stop_IT	522
40.2.36	HAL_TIMEx_ConfigCommutationEvent	522
40.2.37	HAL_TIMEx_ConfigCommutationEvent_IT	522
40.2.38	HAL_TIMEx_ConfigCommutationEvent_DMA	523
40.2.39	HAL_TIMEx_MasterConfigSynchronization	524
40.2.40	HAL_TIMEx_ConfigBreakDeadTime.....	524
40.2.41	HAL_TIMEx_RemapConfig	524
40.2.42	HAL_TIMEx_ChamutationCallback	524
40.2.43	HAL_TIMEx_BreakCallback	525
40.2.44	TIMEEx_DMACommationCplt	525
40.2.45	HAL_TIMEx_HallSensor_GetState	525
40.3	TIMEx Firmware driver defines	525
40.3.1	TIMEx	525
41	HAL TSC Generic Driver	526
41.1	TSC Firmware driver registers structures.....	526
41.1.1	TSC_InitTypeDef	526

41.1.2	TSC_IOConfigTypeDef.....	527
41.1.3	TSC_HandleTypeDef	527
41.2	TSC Firmware driver API description	528
41.2.1	TSC specific features	528
41.2.2	How to use this driver	528
41.2.3	Initialization and de-initialization functions	529
41.2.4	Peripheral Control functions	529
41.2.5	State functions.....	529
41.2.6	HAL_TSC_Init.....	529
41.2.7	HAL_TSC_DelInit.....	529
41.2.8	HAL_TSC_MspInit.....	530
41.2.9	HAL_TSC_MspDeInit	530
41.2.10	HAL_TSC_Start.....	530
41.2.11	HAL_TSC_Start_IT.....	530
41.2.12	HAL_TSC_Stop	530
41.2.13	HAL_TSC_Stop_IT	530
41.2.14	HAL_TSC_GroupGetStatus	531
41.2.15	HAL_TSC_GroupGetValue	531
41.2.16	HAL_TSC_IOConfig	531
41.2.17	HAL_TSC_IODischarge	531
41.2.18	HAL_TSC_GetState	532
41.2.19	HAL_TSC_PollForAcquisition	532
41.2.20	HAL_TSC_IRQHandler	532
41.2.21	HAL_TSC_ConvCpltCallback.....	532
41.2.22	HAL_TSC_ErrorCallback.....	532
41.3	TSC Firmware driver defines.....	533
41.3.1	TSC.....	533
42	HAL UART Generic Driver.....	542
42.1	UART Firmware driver registers structures	542
42.1.1	UART_InitTypeDef	542
42.1.2	UART_AdvFeatureInitTypeDef.....	543
42.1.3	UART_HandleTypeDef.....	544
42.2	UART Firmware driver API description	545
42.2.1	How to use this driver	545
42.2.2	Initialization and Configuration functions	547
42.2.3	IO operation functions	548
42.2.4	Peripheral Control functions	548
42.2.5	Peripheral State and Error functions	548

42.2.6	HAL_UART_Init	549
42.2.7	HAL_HalfDuplex_Init	549
42.2.8	HAL_MultiProcessor_Init	549
42.2.9	HAL_UART_DelInit	549
42.2.10	HAL_UART_MsplInit	550
42.2.11	HAL_UART_MspDelInit	550
42.2.12	HAL_UART_Transmit	550
42.2.13	HAL_UART_Receive	550
42.2.14	HAL_UART_Transmit_IT	550
42.2.15	HAL_UART_Receive_IT	551
42.2.16	HAL_UART_Transmit_DMA	551
42.2.17	HAL_UART_Receive_DMA	551
42.2.18	HAL_UART_DMAPause	551
42.2.19	HAL_UART_DMAResume	551
42.2.20	HAL_UART_DMAStop	552
42.2.21	HAL_UART_TxCpltCallback	552
42.2.22	HAL_UART_TxHalfCpltCallback	552
42.2.23	HAL_UART_RxCpltCallback	552
42.2.24	HAL_UART_RxHalfCpltCallback	552
42.2.25	HAL_UART_ErrorCallback	553
42.2.26	HAL_UART_IRQHandler	553
42.2.27	HAL_MultiProcessor_EnableMuteMode	553
42.2.28	HAL_MultiProcessor_DisableMuteMode	553
42.2.29	HAL_MultiProcessor_EnterMuteMode	553
42.2.30	HAL_HalfDuplex_EnableTransmitter	554
42.2.31	HAL_HalfDuplex_EnableReceiver	554
42.2.32	HAL_UART_GetState	554
42.2.33	HAL_UART_GetError	554
42.3	UART Firmware driver defines	554
42.3.1	UART	554
43	HAL UART Extension Driver	570
43.1	UARTEX Firmware driver registers structures	570
43.1.1	UART_WakeUpTypeDef	570
43.2	UARTEX Firmware driver API description	570
43.2.1	UART peripheral extended features	570
43.2.2	Initialization and Configuration functions	570
43.2.3	IO operation function	571
43.2.4	Peripheral Control function	571

43.2.5	HAL_RS485Ex_Init.....	572
43.2.6	HAL_LIN_Init	572
43.2.7	HAL_UART_IRQHandler.....	573
43.2.8	HAL_UARTEx_WakeupCallback	573
43.2.9	HAL_UARTEx_StopModeWakeUpSourceConfig	573
43.2.10	HAL_UARTEx_EnableStopMode.....	573
43.2.11	HAL_UARTEx_DisableStopMode	573
43.2.12	HAL_MultiProcessorEx_AddressLength_Set.....	574
43.2.13	HAL_LIN_SendBreak	574
43.3	UARTEx Firmware driver defines	574
43.3.1	UARTEx.....	574
44	HAL USART Generic Driver	576
44.1	USART Firmware driver registers structures	576
44.1.1	USART_InitTypeDef	576
44.1.2	USART_HandleTypeDef	577
44.2	USART Firmware driver API description	578
44.2.1	How to use this driver	578
44.2.2	Initialization and Configuration functions	580
44.2.3	IO operation functions	580
44.2.4	Peripheral State and Error functions	582
44.2.5	HAL_USART_Init.....	582
44.2.6	HAL_USART_DeInit	582
44.2.7	HAL_USART_MspInit.....	582
44.2.8	HAL_USART_MspDeInit	582
44.2.9	HAL_USART_Transmit	582
44.2.10	HAL_USART_Receive	583
44.2.11	HAL_USART_TransmitReceive	583
44.2.12	HAL_USART_Transmit_IT	583
44.2.13	HAL_USART_Receive_IT	584
44.2.14	HAL_USART_TransmitReceive_IT	584
44.2.15	HAL_USART_Transmit_DMA	584
44.2.16	HAL_USART_Receive_DMA	584
44.2.17	HAL_USART_TransmitReceive_DMA	585
44.2.18	HAL_USART_DMAPause	585
44.2.19	HAL_USART_DMAResume	585
44.2.20	HAL_USART_DMAStop	585
44.2.21	HAL_USART_IRQHandler	586
44.2.22	HAL_USART_TxCpltCallback	586

44.2.23	HAL_USART_TxHalfCpltCallback	586
44.2.24	HAL_USART_RxCpltCallback	586
44.2.25	HAL_USART_RxHalfCpltCallback	586
44.2.26	HAL_USART_TxRxCpltCallback	586
44.2.27	HAL_USART_ErrorCallback	587
44.2.28	HAL_USART_GetState	587
44.2.29	HAL_USART_GetError	587
44.3	USART Firmware driver defines	587
44.3.1	USART	587
45	HAL USART Extension Driver	596
45.1	USARTEx Firmware driver defines	596
45.1.1	USARTEx	596
46	HAL WWDG Generic Driver	597
46.1	WWDG Firmware driver registers structures	597
46.1.1	WWDG_InitTypeDef	597
46.1.2	WWDG_HandleTypeDef	597
46.2	WWDG Firmware driver API description	598
46.2.1	WWDG specific features	598
46.2.2	How to use this driver	598
46.2.3	Initialization and de-initialization functions	598
46.2.4	IO operation functions	599
46.2.5	Peripheral State functions	599
46.2.6	HAL_WWDG_Init	599
46.2.7	HAL_WWDG_DeInit	599
46.2.8	HAL_WWDG_MspInit	600
46.2.9	HAL_WWDG_MspDeInit	600
46.2.10	HAL_WWDG_Start	600
46.2.11	HAL_WWDG_Start_IT	600
46.2.12	HAL_WWDG_Refresh	600
46.2.13	HAL_WWDG_IRQHandler	601
46.2.14	HAL_WWDG_WakeupCallback	601
46.2.15	HAL_WWDG_GetState	601
46.3	WWDG Firmware driver defines	601
46.3.1	WWDG	601
47	FAQs	605
48	Revision history	609

List of tables

Table 1: Acronyms and definitions	34
Table 2: HAL drivers files	36
Table 3: User-application files	37
Table 4: APis classification	42
Table 5: List of devices supported by HAL drivers	42
Table 6: HAL API naming rules	44
Table 7: Macros handling interrupts and specific clock configurations	45
Table 8: Callback functions	47
Table 9: HAL generic APIs	47
Table 10: HAL extension APIs	48
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	61
Table 15: Timeout values	65
Table 16: COMP Inputs for STM32F05xx, STM32F07x and STM32F09x devices	133
Table 17: COMP outputs for STM32F05xx, STM32F07x and STM32F09x devices	134
Table 18: Redirection of COMP outputs to embedded timers for STM32F05xx, STM32F07x and STM32F09x devices	134
Table 19: IRDA frame formats	260
Table 20: Number of wait states (WS) according to system clock (SYSCLK) frequency	312
Table 21: USART frame formats	407
Table 22: Maximum SPI frequency vs data size	447
Table 23: UART frame formats	547
Table 24: USART frame formats	571
Table 25: USART frame formats	580
Table 26: Document revision history	609

List of figures

Figure 1: Example of project template	39
Figure 2: Adding device-specific functions	49
Figure 3: Adding family-specific functions	49
Figure 4: Adding new peripherals	50
Figure 5: Updating existing APIs	50
Figure 6: File inclusion model	51
Figure 7: HAL driver model	59

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

Acronym	Definition
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">• relocate the vector table in internal SRAM.
<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">• the call to HAL_Init()• assert_failed() implementation• system clock configuration• peripheral HAL initialization and user application code.

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

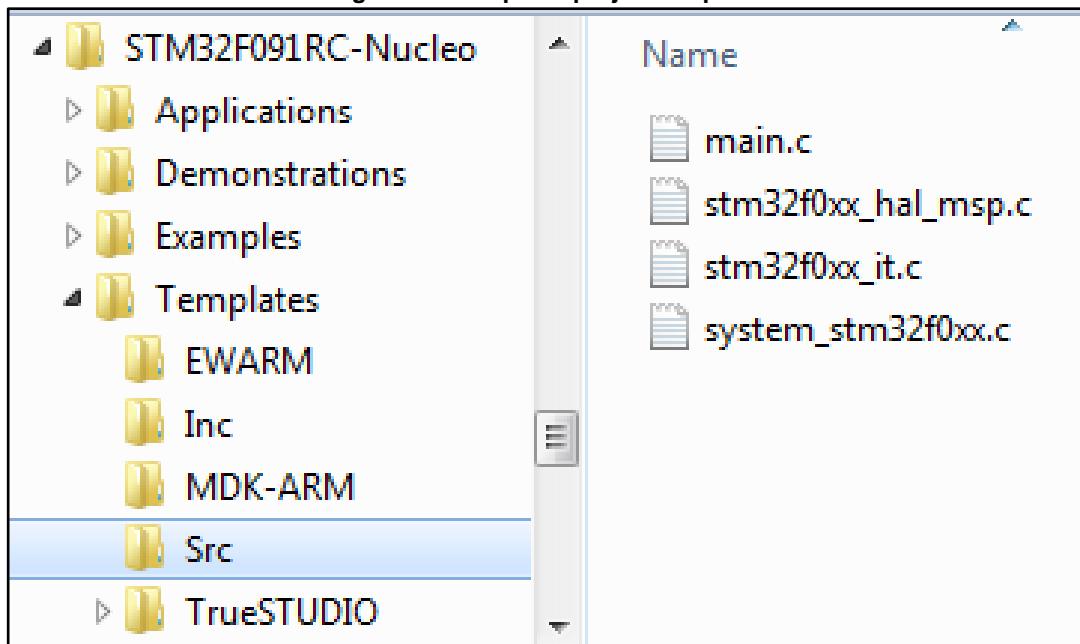
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Their characteristics are the following:

- It contains sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows to configure the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
 - HAL is initialized
 - SysTick ISR implemented for HAL_Delay()
 - System clock configured with the maximum frequency of the device



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

Figure 1: Example of project template



2.2 HAL data structures

Each HAL driver can contain the following data structures:

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

2.2.1 Peripheral handle structures

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

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

The peripheral handle is used for the following purposes:

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

An example of peripheral structure is shown below:

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

```

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

```



1) The multi-instance feature implies that all the APIs used in the application are re-entrant and avoid using global variables because subroutines can fail to be re-entrant if they rely on a global variable to remain unchanged but that variable is modified when the subroutine is recursively invoked. For this reason, the following rules are respected:

- Re-entrant code does not hold any static (or global) non-constant data: re-entrant functions can work with global data. For example, a re-entrant interrupt service routine can grab a piece of hardware status to work with (e.g. serial port read buffer) which is not only global, but volatile. Still, typical use of static variables and global data is not advised, in the sense that only atomic read-modify-write instructions should be used in these variables. It should not be possible for an interrupt or signal to occur during the execution of such an instruction.
- Reentrant code does not modify its own code.



2) When a peripheral can manage several processes simultaneously using the DMA (full duplex case), the DMA interface handle for each process is added in the PPP_HandleTypeDef.



3) For the shared and system peripherals, no handle or instance object is used. The peripherals concerned by this exception are the following:

- GPIO
- SYSTICK
- NVIC
- PWR
- RCC
- FLASH.

2.2.2 Initialization and configuration structure

These structures are defined in the generic driver header file when it is common to all part numbers. When they can change from one part number to another, the structures are defined in the extension header file for each part number.

```

typedef struct
{
    uint32_t BaudRate; /*!< This member configures the UART communication baudrate.*/
    uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received
    in a frame.*/
    uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
    uint32_t Parity; /*!< Specifies the parity mode.*/
    uint32_t Mode; /*!< Specifies whether the Receive or Transmit mode is enabled or
    disabled.*/
    uint32_t HwFlowCtl; /*!< Specifies whether the hardware flow control mode is enabled
    or disabled.*/
    uint32_t OverSampling; /*!< Specifies whether the Over sampling 8 is enabled or
    disabled.*/
}

```

```
disabled,
to achieve higher speed (up to fPCLK/8).*/
}UART_InitTypeDef;
```



The config structure is used to initialize the sub-modules or sub-instances. See below example:

```
HAL_ADC_ConfigChannel (ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef* sConfig)
```

2.2.3 Specific process structures

The specific process structures are used for specific process (common APIs). They are defined in the generic driver header file.

Example:

```
HAL_PPP_Process (PPP_HandleTypeDef* hadc, PPP_ProcessConfig* sConfig)
```

2.3 API classification

The HAL APIs are classified into three categories:

- **Generic APIs:** common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL drivers files of all STM32 microcontrollers.

```
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_DeInit(ADC_HandleTypeDef *hadc);
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop_IT(ADC_HandleTypeDef* hadc);
void HAL_ADC_IRQHandler(ADC_HandleTypeDef* hadc);
```

- **Extension APIs:** This set of API is divided into two sub-categories :
 - **Family specific APIs:** APIs applying to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

```
HAL_StatusTypeDef HAL_ADCEX_Calibration_Start(ADC_HandleTypeDef* hadc, uint32_t SingleDiff);
uint32_t HAL_ADCEX_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t SingleDiff);
```

- **Device part number specific APIs:** These APIs are implemented in the extension file and delimited by specific define statements relative to a given part number.

```
#if defined(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 ⁽¹⁾
Family specific APIs			X
Device specific APIs			X

Notes:

⁽¹⁾In some cases, the implementation for a specific device part number may change . In this case the generic API is declared as weak function in the extension file. The API is implemented again to overwrite the default function



Family specific APIs are only related to a given family. This means that if a specific API is implemented in another family, and the arguments of this latter family are different, additional structures and arguments might need to be added.



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

2.4 Devices supported by HAL drivers

Table 5: List of devices supported by HAL drivers

IP/Module	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_com_p.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

IP/Module	STM32F030x6	STM32F030x8	STM32F030x6	STM32F070x8	STM32F030xC	STM32F031x6	STM32F051x8	STM32F071xB	STM32F091xC	STM32F042x6	STM32F072xB	STM32F048xx	STM32F058xx	STM32F078xx	STM32F098xx
stm32f0xx_hal_dma.c	Yes														
stm32f0xx_hal_flash.c	Yes														
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														

IP/Module	STM32F030x6	STM32F030x8	STM32F070x6	STM32F070x8	STM32F030xC	STM32F031x6	STM32F051x8	STM32F071xB	STM32F091xC	STM32F042x6	STM32F072xB	STM32F048xx	STM32F058xx	STM32F078xx	STM32F098xx
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
File names	<i>stm32f0xx_hal_ppp (c/h)</i>	<i>stm32f0xx_hal_ppp_ex (c/h)</i>	<i>stm32f0xx_hal_ppp_ex (c/h)</i>
Module name	<i>HAL_PPP_MODULE</i>		
Function name	<i>HAL_PPP_Function</i> <i>HAL_PPP_FeatureFunction_MODE</i>	<i>HAL_PPPEx_Function</i> <i>HAL_PPPEx_FeatureFunction_MODE</i>	<i>HAL_PPPEx_Function</i> <i>HAL_PPPEx_FeatureFunction_MODE</i>
Handle name	<i>PPP_HandleTypeDef</i>	NA	NA
Init structure name	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
Enum name	<i>HAL_PPP_StructnameTypeDef</i>	NA	NA

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the 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

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

2.6 HAL generic APIs

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

- **Initialization and de-initialization functions:** HAL_PPP_Init(), HAL_PPP_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>	HAL_ADC_Init()	This function initializes the peripheral and configures the low-level resources (clocks, GPIO, AF..)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
<i>IO operation group</i>	HAL_ADC_Start()	This function starts ADC conversions when the polling method is used

Function Group	Common API Name	Description
	<i>HAL_ADC_Stop()</i>	This function stops ADC conversions when the polling method is used
	<i>HAL_ADC_PollForConversion()</i>	This function allows waiting for the end of conversions when the polling method is used. In this case, a timeout value is specified by the user according to the application.
	<i>HAL_ADC_Start_IT()</i>	This function starts ADC conversions when the interrupt method is used
	<i>HAL_ADC_Stop_IT()</i>	This function stops ADC conversions when the interrupt method is used
	<i>HAL_ADC_IRQHandler()</i>	This function handles ADC interrupt requests
	<i>HAL_ADC_ConvCpltCallback()</i>	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	<i>HAL_ADC_ErrorCallback()</i>	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
<i>Control group</i>	<i>HAL_ADC_ConfigChannel()</i>	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	<i>HAL_ADC_AnalogWDGConfig</i>	This function configures the analog watchdog for the selected ADC
<i>State and Errors group</i>	<i>HAL_ADC_GetState()</i>	This function allows getting in runtime the peripheral and the data flow states.
	<i>HAL_ADC_GetError()</i>	This function allows getting in runtime the error that occurred during IT routine

2.7 HAL extension APIs

2.7.1 HAL extension model overview

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

The extension model consists of an additional file, `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

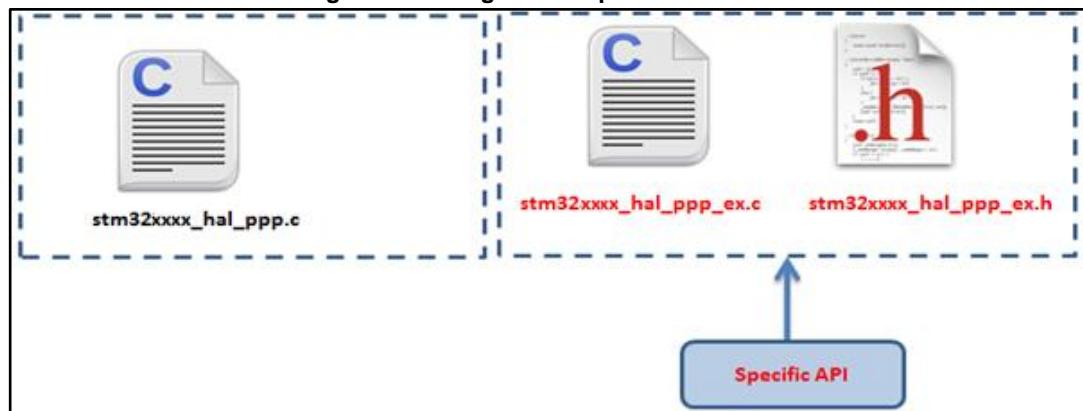
2.7.2 HAL extension model cases

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



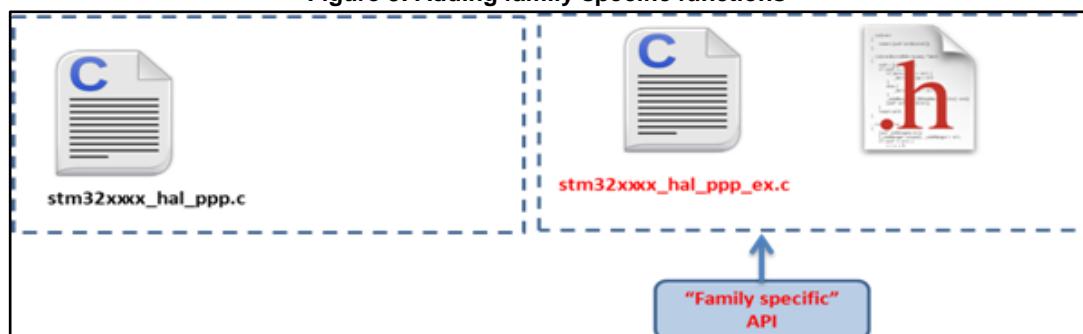
Example: `stm32f0xx_hal_rcc_ex.c/h`

```
#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_CRSConfig(RCC 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_PPPEx_Function()`.

Figure 3: Adding family-specific functions

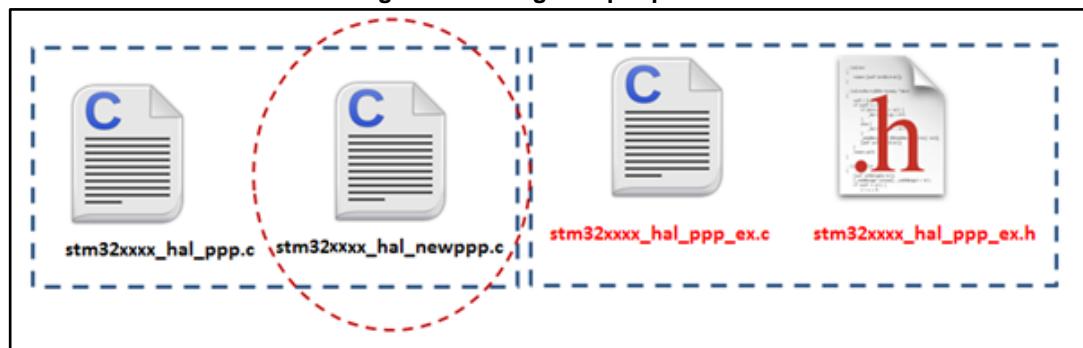


Case3 : Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in `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

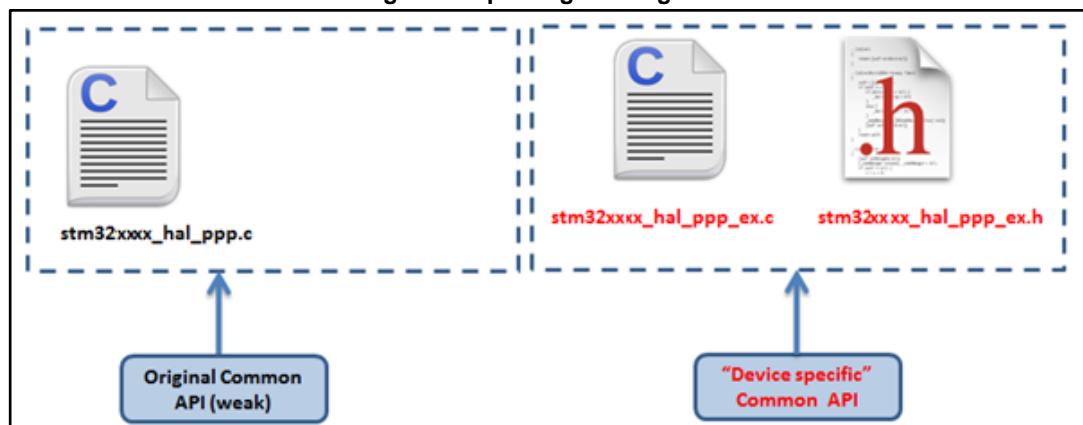


Example: `stm32f0xx_hal_lcd.c/h`

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:

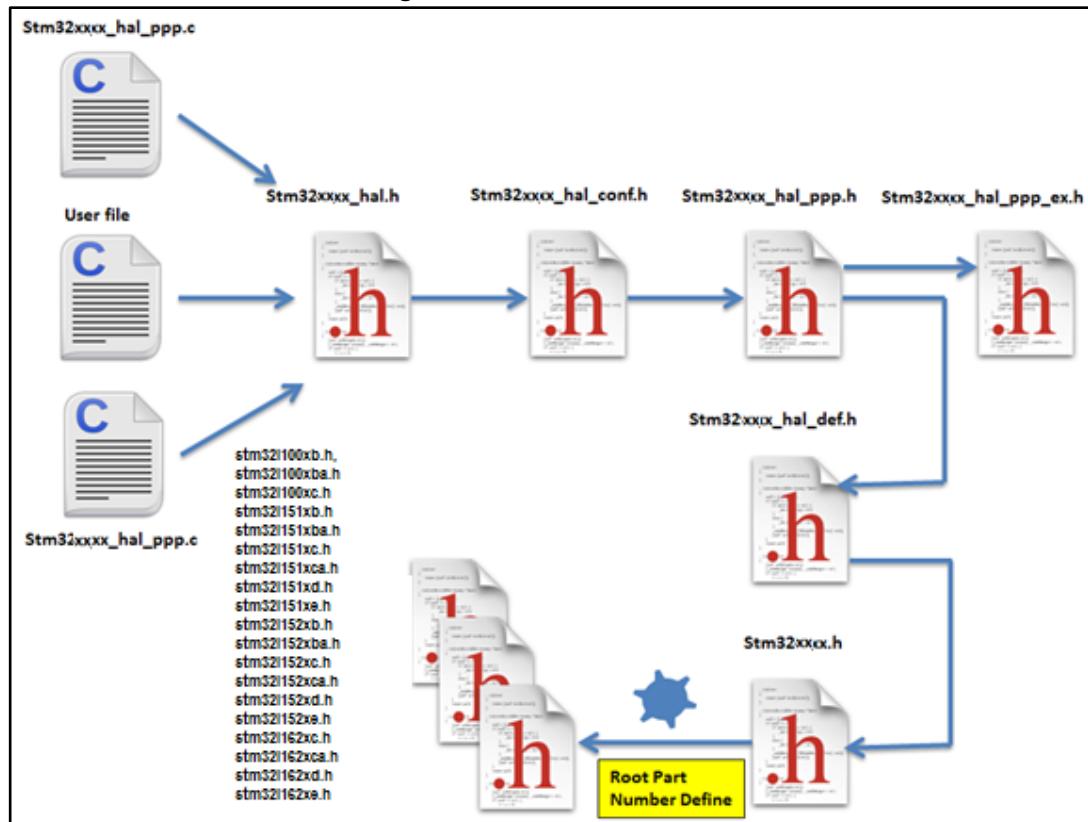
```
#if defined (STM32F072xB)
typedef struct
{
(...)

}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 macro**
 - Macro defining NULL and HAL_MAX_DELAY

```
#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)
LSE_STARTUP_TIMEOUT	Timeout for LSE start up, expressed in ms	5000
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)
HSI48_VALUE	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)
LSI_VALUE	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)
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	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_RCCE_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)`.

Additional RCC HAL driver functions are available:

- `HAL_RCC_DelInit()` 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	<p>Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]</p>
Mode	<p>Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are:</p> <ul style="list-style-type: none"> • <u>GPIO mode</u> <ul style="list-style-type: none"> – GPIO_MODE_INPUT : Input Floating – GPIO_MODE_OUTPUT_PP : Output Push Pull – GPIO_MODE_OUTPUT_OD : Output Open Drain – GPIO_MODE_AF_PP : Alternate Function Push Pull – GPIO_MODE_AF_OD : Alternate Function Open Drain – GPIO_MODE_ANALOG : Analog mode • <u>External Interrupt Mode</u> <ul style="list-style-type: none"> – GPIO_MODE_IT_RISING : Rising edge trigger detection – GPIO_MODE_IT_FALLING : Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection • <u>External Event Mode</u> <ul style="list-style-type: none"> – GPIO_MODE_EVT_RISING : Rising edge trigger detection – GPIO_MODE_EVT_FALLING : Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	<p>Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN</p>
Speed	<p>Specifies the speed for the selected pins Possible values are: GPIO_SPEED_LOW GPIO_SPEED_MEDIUM GPIO_SPEED_HIGH</p>
Alternate	<p>Peripheral to be connected to the selected pins. Possible values: GPIO_AFx PPP, where AFx: is the alternate function index PPP: is the peripheral instance Example: use GPIO_AF1_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.</p> <p> Refer to the “Alternate function mapping” table in the datasheets for the detailed description of the system and peripheral I/O alternate functions.</p>

Please find below typical GPIO configuration examples:

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

```
GPIO_InitStruct.Pin = GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_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) /*!<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>
__HAL_PPP_EXTI_DISABLE_IT	Disables a given EXTI line. Example: <code>__HAL_PWR_PVD_EXTI_DISABLE_IT()</code>
__HAL_PPP_EXTI_GET_FLAG	Gets a given EXTI line interrupt flag pending bit status. Example: <code>__HAL_PWR_PVD_EXTI_GET_FLAG()</code>
__HAL_PPP_EXTI_CLEAR_FLAG	Clears a given EXTI line interrupt flag pending bit. Example; <code>__HAL_PWR_PVD_EXTI_CLEAR_FLAG()</code>
__HAL_PPP_EXTI_GENERATE_SWIT	Generates a software interrupt for a given EXTI line. Example: <code>__HAL_PWR_PVD_EXTI_GENERATE_SWIT()</code>
__HAL_PPP_EXTI_ENABLE_EVENT	Enables event on a given EXTI Line Example: <code>__HAL_PWR_PVD_EXTI_ENABLE_EVENT()</code>
__HAL_PPP_EXTI_DISABLE_EVENT	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”).



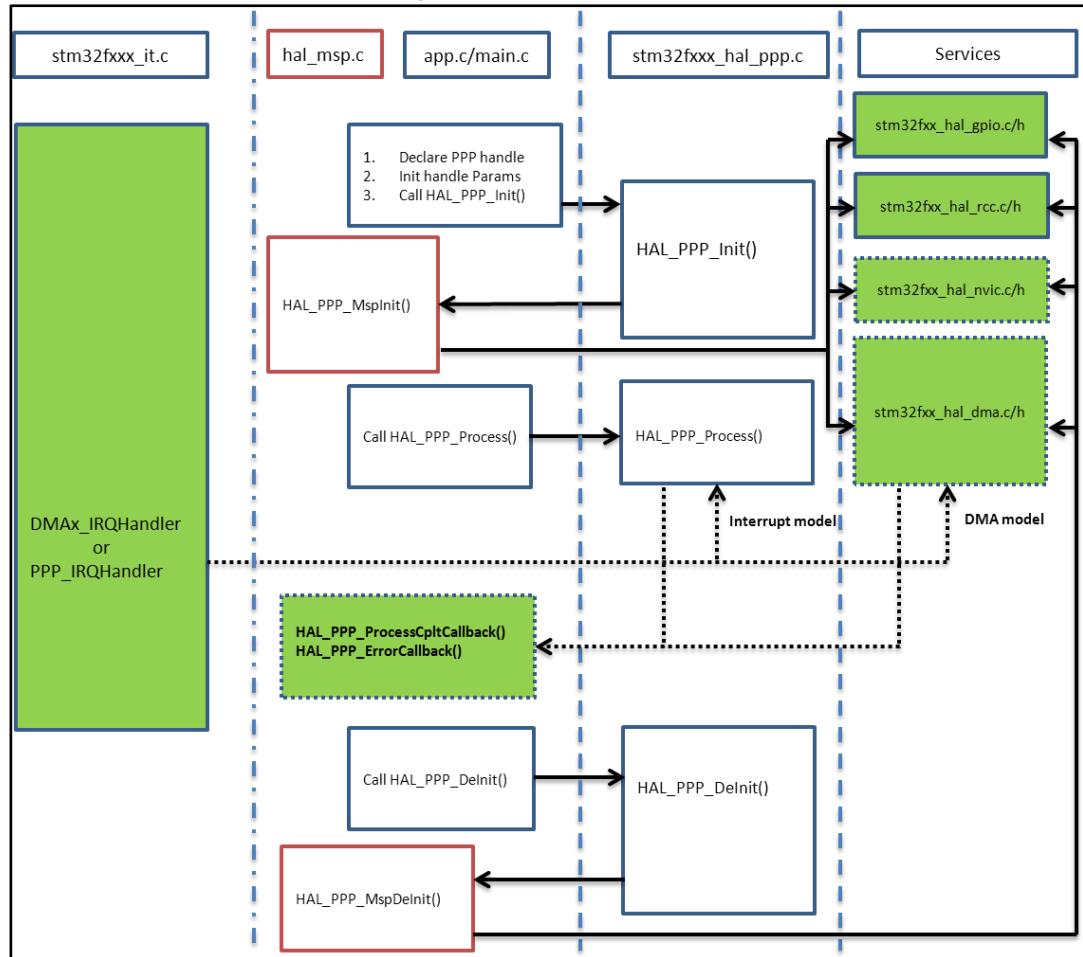
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

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

Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR.

This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be blocked.

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

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

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

peripheral and other peripherals initialization and de-Initialization are kept in the global *HAL_MspInit()* and the *HAL_MspDeInit()*.

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

2.12.3 HAL IO operation process

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

- Polling mode
- Interrupt mode
- DMA mode

2.12.3.1 Polling mode

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

The example below shows the typical polling mode processing sequence :

```
HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t pData,
int16_tSize, uint32_tTimeout)
{
if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(...) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(...)
return 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_ProcCpltCallback()*: the callback relative to the process completion.
- *__weak HAL_PPP_ProcErrorCallback()*: 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.c file:

```
extern UART_HandleTypeDef UartHandle;
void USART1_IRQHandler(void)
{
HAL_UART_IRQHandler(&UartHandle);
}
```

2.12.3.3 DMA mode

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

- *HAL_PPP_Process_DMA()*: launch the process
- *HAL_PPP_DMA_IRQHandler()*: the DMA interruption used by the PPP peripheral
- *__weak HAL_PPP_ProcessCpltCallback()*: the callback relative to the process completion.
- *__weak HAL_PPP_ErrorCpltCallback()*: the callback relative to the process Error.

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *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_TxErrorCallback(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) ⁽¹⁾	Timeout in ms
HAL_MAX_DELAY	Infinite poll till process is successful

Notes:

⁽¹⁾HAL_MAX_DELAY is defined in the stm32fxxx_hal_def.h as 0xFFFFFFFF

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

Example of fixed timeout:

```

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

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

```

HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hPPP, uint32_t Timeout)
{
(...)
timeout = HAL_GetTick() + Timeout;
(...)
while(ProcessOngoing)
{
```

```

(...)  

if(Timeout != HAL_MAX_DELAY)  

{  

if(HAL_GetTick() >= timeout)  

{  

/* Process unlocked */  

__HAL_UNLOCK(hppp);  

hppp->State= HAL_PPP_STATE_TIMEOUT;  

return hppp->State;  

}  

}  

(...)  

}

```

2.12.4.2 Error management

The HAL drivers implement a check for the following items:

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

```

HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pdata, uint32  

Size)
{
if ((pData == NULL) || (Size == 0))
{
return HAL_ERROR;
}
}

```

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

```

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

```

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

```

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

```

When an error occurs during a peripheral process, *HAL_PPP_Process()* returns with a *HAL_ERROR* status. The HAL PPP driver implements the *HAL_PPP_GetError()* to allow retrieving the origin of the error.

```

HAL_PPP_ErrorTypeDef HAL_PPP_GetError (PPP_HandleTypeDef *hppp);

```

In all peripheral handles, a *HAL_PPP_ErrorTypeDef* is defined and used to store the last error code.

```

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

```

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

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

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

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

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

2.12.4.3 Run-time checking

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

To enable the run-time checking, use the *assert_param* macro, and leave the define **USE_FULL_ASSERT** uncommented in *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);
#endif
```

```
#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

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.

This section contains the following APIs:

- [**HAL_Init\(\)**](#)
- [**HAL_DelInit\(\)**](#)
- [**HAL_MspInit\(\)**](#)
- [**HAL_MspDelInit\(\)**](#)
- [**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

This section contains the following APIs:

- [*HAL_IncTick\(\)*](#)
- [*HAL_GetTick\(\)*](#)
- [*HAL_Delay\(\)*](#)
- [*HAL_SuspendTick\(\)*](#)
- [*HAL_ResumeTick\(\)*](#)
- [*HAL_GetHalVersion\(\)*](#)
- [*HAL_GetREVID\(\)*](#)
- [*HAL_GetDEVID\(\)*](#)
- [*HAL_DBGMCU_EnableDBGStopMode\(\)*](#)
- [*HAL_DBGMCU_DisableDBGStopMode\(\)*](#)
- [*HAL_DBGMCU_EnableDBGStandbyMode\(\)*](#)
- [*HAL_DBGMCU_DisableDBGStandbyMode\(\)*](#)

3.1.4 HAL_Init

Function Name	HAL_StatusTypeDef HAL_Init (void)
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">• HAL status
Notes	<ul style="list-style-type: none">• This function is called at the beginning of program after reset and before the clock configuration• The time base configuration is based on 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.

3.1.5 HAL_DeInit

Function Name	HAL_StatusTypeDef HAL_DeInit (void)
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">• HAL status
Notes	<ul style="list-style-type: none">• This function is optional.

3.1.6 HAL_MspInit

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

3.1.7 HAL_MspDeInit

Function Name	void HAL_MspDeInit (void)
---------------	-----------------------------------

	Function Description	Deinitializes the MSP.
	Return values	<ul style="list-style-type: none"> None
3.1.8	HAL_InitTick	
	Function Name	HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)
	Function Description	This function configures the source of the time base.
	Parameters	<ul style="list-style-type: none"> TickPriority: Tick interrupt priority.
	Return values	<ul style="list-style-type: none"> HAL status
	Notes	<ul style="list-style-type: none"> This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig(). In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, The 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	void HAL_IncTick (void)
	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"> None
	Notes	<ul style="list-style-type: none"> In the default implementation, this variable is incremented each 1ms in Systick ISR. This function is declared as __weak to be overwritten in case of other implementations in user file.
3.1.10	HAL_GetTick	
	Function Name	uint32_t HAL_GetTick (void)
	Function Description	Provides a tick value in millisecond.
	Return values	<ul style="list-style-type: none"> tick value
	Notes	<ul style="list-style-type: none"> This function is declared as __weak to be overwritten in case of other implementations in user file.
3.1.11	HAL_Delay	
	Function Name	void HAL_Delay (__IO uint32_t Delay)
	Function Description	This function provides accurate delay (in milliseconds) based on variable incremented.
	Parameters	<ul style="list-style-type: none"> Delay: specifies the delay time length, in milliseconds.
	Return values	<ul style="list-style-type: none"> None

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

3.1.12 HAL_SuspendTick

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

3.1.13 HAL_ResumeTick

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

3.1.14 HAL_GetHalVersion

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

3.1.15 HAL_GetREVID

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

3.1.16 HAL_GetDEVID

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

3.1.17 HAL_DBGMCU_EnableDBGStopMode

Function Name **void HAL_DBGMCU_EnableDBGStopMode (void)**

Function Description Enable the Debug Module during STOP mode.

Return values • None

3.1.18 HAL_DBGMCU_DisableDBGStopMode

Function Name **void HAL_DBGMCU_DisableDBGStopMode (void)**

Function Description Disable the Debug Module during STOP mode.

Return values • None

3.1.19 HAL_DBGMCU_EnableDBGStandbyMode

Function Name **void HAL_DBGMCU_EnableDBGStandbyMode (void)**

Function Description Enable the Debug Module during STANDBY mode.

Return values • None

3.1.20 HAL_DBGMCU_DisableDBGStandbyMode

Function Name **void HAL_DBGMCU_DisableDBGStandbyMode (void)**

Function Description Disable the Debug Module during STANDBY mode.

Return values • None

3.2 HAL Firmware driver defines

3.2.1 HAL

CAN Error Code

HAL_CAN_ERROR_NONE No error

HAL_CAN_ERROR_EWG EWG error

HAL_CAN_ERROR_EPV EPV error

HAL_CAN_ERROR_BOF BOF error

HAL_CAN_ERROR_STF Stuff error

HAL_CAN_ERROR_FOR Form error

HAL_CAN_ERROR_ACK Acknowledgment error

HAL_CAN_ERROR_BR Bit recessive

HAL_CAN_ERROR_BD LEC dominant

HAL_CAN_ERROR_CRC LEC transfer error

HAL Exported Macros

_HAL_SYSCFG_FASTMODEPLUS_ENABLE **Description:**

- Fast-mode Plus driving capability enable/disable macros.

Parameters:

- _FASTMODEPLUS_: This parameter can be a value of

_HAL_SYSCFG_FASTMODEPLUS_DISABLE

HAL Freeze Unfreeze Peripherals

_HAL_FREEZE_CAN_DBGMCU
_HAL_UNFREEZE_CAN_DBGMCU
_HAL_DBGMCU_FREEZE_RTC
_HAL_DBGMCU_UNFREEZE_RTC
_HAL_DBGMCU_FREEZE_I2C1_TIMEOUT
_HAL_DBGMCU_UNFREEZE_I2C1_TIMEOUT
_HAL_DBGMCU_FREEZE_IWDG
_HAL_DBGMCU_UNFREEZE_IWDG
_HAL_DBGMCU_FREEZE_WWDG
_HAL_DBGMCU_UNFREEZE_WWDG
_HAL_DBGMCU_FREEZE_TIM2
_HAL_DBGMCU_UNFREEZE_TIM2
_HAL_DBGMCU_FREEZE_TIM3
_HAL_DBGMCU_UNFREEZE_TIM3
_HAL_DBGMCU_FREEZE_TIM6
_HAL_DBGMCU_UNFREEZE_TIM6
_HAL_DBGMCU_FREEZE_TIM7
_HAL_DBGMCU_UNFREEZE_TIM7
_HAL_DBGMCU_FREEZE_TIM14
_HAL_DBGMCU_UNFREEZE_TIM14
_HAL_DBGMCU_FREEZE_TIM1
_HAL_DBGMCU_UNFREEZE_TIM1
_HAL_DBGMCU_FREEZE_TIM15
_HAL_DBGMCU_UNFREEZE_TIM15
_HAL_DBGMCU_FREEZE_TIM16
_HAL_DBGMCU_UNFREEZE_TIM16
_HAL_DBGMCU_FREEZE_TIM17
_HAL_DBGMCU_UNFREEZE_TIM17

HAL IRDA Enveloppe Selection

HAL_SYSCFG_IRDA_ENV_SEL_TIM16
HAL_SYSCFG_IRDA_ENV_SEL_USART1
HAL_SYSCFG_IRDA_ENV_SEL_USART4

HAL ISR Wrapper

HAL_SYSCFG_ITLINE0	Internal define for macro handling
HAL_SYSCFG_ITLINE1	Internal define for macro handling
HAL_SYSCFG_ITLINE2	Internal define for macro handling
HAL_SYSCFG_ITLINE3	Internal define for macro handling
HAL_SYSCFG_ITLINE4	Internal define for macro handling
HAL_SYSCFG_ITLINE5	Internal define for macro handling
HAL_SYSCFG_ITLINE6	Internal define for macro handling
HAL_SYSCFG_ITLINE7	Internal define for macro handling
HAL_SYSCFG_ITLINE8	Internal define for macro handling
HAL_SYSCFG_ITLINE9	Internal define for macro handling
HAL_SYSCFG_ITLINE10	Internal define for macro handling
HAL_SYSCFG_ITLINE11	Internal define for macro handling
HAL_SYSCFG_ITLINE12	Internal define for macro handling
HAL_SYSCFG_ITLINE13	Internal define for macro handling
HAL_SYSCFG_ITLINE14	Internal define for macro handling
HAL_SYSCFG_ITLINE15	Internal define for macro handling
HAL_SYSCFG_ITLINE16	Internal define for macro handling
HAL_SYSCFG_ITLINE17	Internal define for macro handling
HAL_SYSCFG_ITLINE18	Internal define for macro handling
HAL_SYSCFG_ITLINE19	Internal define for macro handling
HAL_SYSCFG_ITLINE20	Internal define for macro handling
HAL_SYSCFG_ITLINE21	Internal define for macro handling
HAL_SYSCFG_ITLINE22	Internal define for macro handling
HAL_SYSCFG_ITLINE23	Internal define for macro handling
HAL_SYSCFG_ITLINE24	Internal define for macro handling
HAL_SYSCFG_ITLINE25	Internal define for macro handling
HAL_SYSCFG_ITLINE26	Internal define for macro handling
HAL_SYSCFG_ITLINE27	Internal define for macro handling
HAL_SYSCFG_ITLINE28	Internal define for macro handling
HAL_SYSCFG_ITLINE29	Internal define for macro handling
HAL_SYSCFG_ITLINE30	Internal define for macro handling
HAL_SYSCFG_ITLINE31	Internal define for macro handling
HAL_ITLINE_EWDG	EWDG has expired
HAL_ITLINE_PVDOUT	Power voltage detection Interrupt
HAL_ITLINE_VDDIO2	VDDIO2 Interrupt

HAL_ITLINE_RTC_WAKEUP	RTC WAKEUP -> exti[20] Interrupt
HAL_ITLINE_RTC_TSTAMP	RTC Time Stamp -> exti[19] interrupt
HAL_ITLINE_RTC_ALRA	RTC Alarm -> exti[17] interrupt
HAL_ITLINE_FLASH_ITF	Flash ITF Interrupt
HAL_ITLINE_CRS	CRS Interrupt
HAL_ITLINE_CLK_CTRL	CLK Control Interrupt
HAL_ITLINE_EXTI0	External Interrupt 0
HAL_ITLINE_EXTI1	External Interrupt 1
HAL_ITLINE_EXTI2	External Interrupt 2
HAL_ITLINE_EXTI3	External Interrupt 3
HAL_ITLINE_EXTI4	EXTI4 Interrupt
HAL_ITLINE_EXTI5	EXTI5 Interrupt
HAL_ITLINE_EXTI6	EXTI6 Interrupt
HAL_ITLINE_EXTI7	EXTI7 Interrupt
HAL_ITLINE_EXTI8	EXTI8 Interrupt
HAL_ITLINE_EXTI9	EXTI9 Interrupt
HAL_ITLINE_EXTI10	EXTI10 Interrupt
HAL_ITLINE_EXTI11	EXTI11 Interrupt
HAL_ITLINE_EXTI12	EXTI12 Interrupt
HAL_ITLINE_EXTI13	EXTI13 Interrupt
HAL_ITLINE_EXTI14	EXTI14 Interrupt
HAL_ITLINE_EXTI15	EXTI15 Interrupt
HAL_ITLINE_TSC_EOA	Touch control EOA Interrupt
HAL_ITLINE_TSC_MCE	Touch control MCE Interrupt
HAL_ITLINE_DMA1_CH1	DMA1 Channel 1 Interrupt
HAL_ITLINE_DMA1_CH2	DMA1 Channel 2 Interrupt
HAL_ITLINE_DMA1_CH3	DMA1 Channel 3 Interrupt
HAL_ITLINE_DMA2_CH1	DMA2 Channel 1 Interrupt
HAL_ITLINE_DMA2_CH2	DMA2 Channel 2 Interrupt
HAL_ITLINE_DMA1_CH4	DMA1 Channel 4 Interrupt
HAL_ITLINE_DMA1_CH5	DMA1 Channel 5 Interrupt
HAL_ITLINE_DMA1_CH6	DMA1 Channel 6 Interrupt
HAL_ITLINE_DMA1_CH7	DMA1 Channel 7 Interrupt
HAL_ITLINE_DMA2_CH3	DMA2 Channel 3 Interrupt
HAL_ITLINE_DMA2_CH4	DMA2 Channel 4 Interrupt
HAL_ITLINE_DMA2_CH5	DMA2 Channel 5 Interrupt

HAL_ITLINE_ADC	ADC Interrupt
HAL_ITLINE_COMP1	COMP1 Interrupt -> exti[21]
HAL_ITLINE_COMP2	COMP2 Interrupt -> exti[21]
HAL_ITLINE_TIM1_BRK	TIM1 BRK Interrupt
HAL_ITLINE_TIM1_UPD	TIM1 UPD Interrupt
HAL_ITLINE_TIM1_TRG	TIM1 TRG Interrupt
HAL_ITLINE_TIM1_CCU	TIM1 CCU Interrupt
HAL_ITLINE_TIM1_CC	TIM1 CC Interrupt
HAL_ITLINE_TIM2	TIM2 Interrupt
HAL_ITLINE_TIM3	TIM3 Interrupt
HAL_ITLINE_DAC	DAC Interrupt
HAL_ITLINE_TIM6	TIM6 Interrupt
HAL_ITLINE_TIM7	TIM7 Interrupt
HAL_ITLINE_TIM14	TIM14 Interrupt
HAL_ITLINE_TIM15	TIM15 Interrupt
HAL_ITLINE_TIM16	TIM16 Interrupt
HAL_ITLINE_TIM17	TIM17 Interrupt
HAL_ITLINE_I2C1	I2C1 Interrupt -> exti[23]
HAL_ITLINE_I2C2	I2C2 Interrupt
HAL_ITLINE_SPI1	I2C1 Interrupt -> exti[23]
HAL_ITLINE_SPI2	SPI1 Interrupt
HAL_ITLINE_USART1	USART1 GLB Interrupt -> exti[25]
HAL_ITLINE_USART2	USART2 GLB Interrupt -> exti[26]
HAL_ITLINE_USART3	USART3 Interrupt
HAL_ITLINE_USART4	USART4 Interrupt
HAL_ITLINE_USART5	USART5 Interrupt
HAL_ITLINE_USART6	USART6 Interrupt
HAL_ITLINE_USART7	USART7 Interrupt
HAL_ITLINE_USART8	USART8 Interrupt
HAL_ITLINE_CAN	CAN Interrupt
HAL_ITLINE_CEC	CEC Interrupt -> exti[27]
HAL ISR wrapper check	
__HAL_GET_PENDING_IT	
HAL state definition	
HAL_SMBUS_STATE_RESET	SMBUS not yet initialized or disabled
HAL_SMBUS_STATE_READY	SMBUS initialized and ready for use

HAL_SMBUS_STATE_BUSY	SMBUS internal process is ongoing
HAL_SMBUS_STATE_MASTER_BUSY_TX	Master Data Transmission process is ongoing
HAL_SMBUS_STATE_MASTER_BUSY_RX	Master Data Reception process is ongoing
HAL_SMBUS_STATE_SLAVE_BUSY_TX	Slave Data Transmission process is ongoing
HAL_SMBUS_STATE_SLAVE_BUSY_RX	Slave Data Reception process is ongoing
HAL_SMBUS_STATE_TIMEOUT	Timeout state
HAL_SMBUS_STATE_ERROR	Reception process is ongoing
HAL_SMBUS_STATE_LISTEN	Address Listen Mode is ongoing

HAL SYSCFG IRDA modulation envelope selection

`_HAL_SYSCFG_IRDA_ENV_SELECTION`
`_HAL_SYSCFG_GET_IRDA_ENV_SELECTION`

HAL SYSCFG Parity check on RAM

`_HAL_SYSCFG_RAM_PARITYCHECK_DISABLE`

Fast-mode Plus on GPIO

`SYSCFG_FASTMODEPLUS_PA9` Enable Fast-mode Plus on PA9
`SYSCFG_FASTMODEPLUS_PA10` Enable Fast-mode Plus on PA10
`SYSCFG_FASTMODEPLUS_PB6` Enable Fast-mode Plus on PB6
`SYSCFG_FASTMODEPLUS_PB7` Enable Fast-mode Plus on PB7
`SYSCFG_FASTMODEPLUS_PB8` Enable Fast-mode Plus on PB8
`SYSCFG_FASTMODEPLUS_PB9` Enable Fast-mode Plus on PB9

4 HAL ADC Generic Driver

4.1 ADC Firmware driver registers structures

4.1.1 ADC_InitTypeDef

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*
- *uint32_t SamplingTimeCommon*

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) has been treated by user software, using function **HAL_ADC_GetValue()**. This feature automatically adapts the ADC conversions trigs 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**
- ***uint32_t ADC_InitTypeDef::SamplingTimeCommon***
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). Note: On STM32F0 devices, the sampling time setting is common to all

channels. On some other STM32 devices, this parameter in channel wise and is located into ADC channel initialization structure. This parameter can be a value of ***ADC_sampling_times*** 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).

4.1.2 ADC_ChannelConfTypeDef

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, number of ranks in the sequence is defined by number of channels enabled, rank of each channel is defined by 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. Caution: Obsolete parameter. Use parameter "SamplingTimeCommon" in ADC initialization structure. If parameter "SamplingTimeCommon" is set to a valid sampling time, parameter "SamplingTime" is discarded. 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

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 Low threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F respectively.

4.1.4 ADC_HandleTypeDef**Data Fields**

- *ADC_TypeDef * Instance*
- *ADC_InitTypeDef Init*
- *DMA_HandleTypeDef * DMA_Handle*
- *HAL_LockTypeDef Lock*
- *__IO uint32_t State*
- *__IO uint32_t ErrorCode*

Field Documentation

- ***ADC_TypeDef* ADC_HandleTypeDef::Instance***
Register base address
- ***ADC_InitTypeDef ADC_HandleTypeDef::Init***
ADC required parameters
- ***DMA_HandleTypeDef* ADC_HandleTypeDef::DMA_Handle***
Pointer DMA Handler
- ***HAL_LockTypeDef ADC_HandleTypeDef::Lock***
ADC locking object
- ***__IO uint32_t ADC_HandleTypeDef::State***
ADC communication state (bitmap of ADC states)

- `_IO uint32_t ADC_HandleTypeDef::ErrorCode`
ADC Error code

4.2 ADC Firmware driver API description

4.2.1 ADC peripheral features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
- Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.
- Programmable sampling time (common for all channels)
- ADC conversion of regular group.
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- ADC calibration
- ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

4.2.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
 - As prerequisite, ADC clock must be configured at RCC top level. Caution: On STM32F0, ADC clock frequency max is 14MHz (refer to device datasheet). Therefore, ADC clock prescaler must be configured in function of ADC clock source frequency to remain below this maximum frequency.
 - Two clock settings are mandatory:
 - ADC clock (core clock, also possibly conversion clock).
 - ADC clock (conversions clock). Two possible clock sources: synchronous clock derived from APB clock or asynchronous clock derived from ADC dedicated HSI RC oscillator 14MHz. If asynchronous clock is selected, 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.
 - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
 - `_HAL_RCC_ADC1_CLK_ENABLE();` (mandatory) HSI14 enable or let under control of ADC: (optional: if asynchronous clock selected)
 - `RCC_OscInitTypeDef RCC_OscInitStruct;`

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

Configuration of ADC, group regular, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ...) using function HAL_ADC_Init().
2. Configure the channels for regular group parameters (channel number, channel rank into sequencer, ..., into regular group) using function HAL_ADC_ConfigChannel().
3. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function HAL_ADC_AnalogWDGConfig().

Execution of ADC conversions

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

- Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() (this function must be implemented in user program)
- Retrieve conversion results using function HAL_ADC_GetValue()
- Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_IT()
- ADC conversion with transfer by DMA:
 - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_DMA()
 - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() or HAL_ADC_ConvHalfCpltCallback() (these functions must be implemented in user program)
 - Conversion results are automatically transferred by DMA into destination variable address.
 - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_DMA()



Callback functions must be implemented in user program:

- HAL_ADC_ErrorCallback()
- HAL_ADC_LevelOutOfWindowCallback() (callback of analog watchdog)
- HAL_ADC_ConvCpltCallback()
- HAL_ADC_ConvHalfCpltCallback

Deinitialization of ADC

1. Disable the ADC interface
 - ADC clock can be hard reset and disabled at RCC top level.
 - Hard reset of ADC peripherals using macro __ADCx_FORCE_RESET(), __ADCx_RELEASE_RESET().
 - ADC clock disable using the equivalent macro/functions as configuration step.
 - Example: Into HAL_ADC_MspDelInit() (recommended code location) or with other device clock parameters configuration:
 - RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI14;
 - RCC_OscInitStructure.HSI14State = RCC_HSI14_OFF; (if not used for system clock)
 - HAL_RCC_OscConfig(&RCC_OscInitStructure);
2. ADC pins configuration
 - Disable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
 - Disable the NVIC for ADC using function HAL_NVIC_DisableIRQ(ADCx_IRQn)
4. Optionally, in case of usage of DMA:
 - Deinitialize the DMA using function HAL_DMA_Init().
 - Disable the NVIC for DMA using function HAL_NVIC_DisableIRQ(DMAx_Channelx_IRQn)

4.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the ADC.
- De-initialize the ADC

This section contains the following APIs:

- [*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.

This section contains the following APIs:

- [*HAL_ADC_Start\(\)*](#)
- [*HAL_ADC_Stop\(\)*](#)
- [*HAL_ADC_PollForConversion\(\)*](#)
- [*HAL_ADC_PollForEvent\(\)*](#)
- [*HAL_ADC_Start_IT\(\)*](#)
- [*HAL_ADC_Stop_IT\(\)*](#)
- [*HAL_ADC_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

This section contains the following APIs:

- [*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

This section contains the following APIs:

- [***HAL_ADC_GetState\(\)***](#)
- [***HAL_ADC_GetError\(\)***](#)

4.2.7 HAL_ADC_Init

Function Name	HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef *hadc)
Function Description	Initializes the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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(). • 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". • 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".

4.2.8 HAL_ADC_DeInit

Function Name	HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef *hadc)
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"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- 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 common group is still running.

4.2.9 HAL_ADC_MspInit

Function Name	void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)
Function Description	Initializes the ADC MSP.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • None

4.2.10 HAL_ADC_MspDeInit

Function Name	void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)
Function Description	Deinitializes the ADC MSP.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • None

4.2.11 HAL_ADC_Start

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

4.2.12 HAL_ADC_Stop

Function Name	HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)
Function Description	Stop ADC conversion of regular group, disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • HAL status.

4.2.13 HAL_ADC_PollForConversion

Function Name	HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)
Function Description	Wait for regular group conversion to be completed.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle • Timeout: Timeout value in millisecond.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function, with an exception: if low power feature "LowPowerAutoWait" is enabled, flags are not cleared to not interfere with this feature until data register is read using function HAL_ADC_GetValue(). • This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each conversion (ADC init parameter "EOCSelection" set to ADC_EOC_SINGLE_CONV). In this case, DMA resets the flag EOC and polling cannot be performed on each

conversion. Nevertheless, polling can still be performed on the complete sequence (ADC init parameter "EOCSelection" set to ADC_EOC_SEQ_CONV).

4.2.14 HAL_ADC_PollForEvent

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

4.2.15 HAL_ADC_Start_IT

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

4.2.16 HAL_ADC_Stop_IT

Function Name	<code>HAL_StatusTypeDef HAL_ADC_Stop_IT(ADC_HandleTypeDef * hadc)</code>
Function Description	Stop ADC conversion of regular group, disable interruption of end-of-conversion, disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • HAL status.

4.2.17 HAL_ADC_Start_DMA

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

4.2.18 HAL_ADC_Stop_DMA

Function Name	<code>HAL_StatusTypeDef HAL_ADC_Stop_DMA(ADC_HandleTypeDef * hadc)</code>
Function Description	Stop ADC conversion of regular group, disable ADC DMA transfer, disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • HAL status.

4.2.19 HAL_ADC_GetValue

Function Name	uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)
Function Description	Get ADC regular group conversion result.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• Converted value
Notes	<ul style="list-style-type: none">• Reading DR register automatically clears EOC (end of conversion of regular group) flag. Additionally, this function clears EOS (end of sequence of regular group) flag, in case of the end of the sequence is reached.

4.2.20 HAL_ADC_IRQHandler

Function Name	void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)
Function Description	Handles ADC interrupt request.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• None

4.2.21 HAL_ADC_ConvCpltCallback

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

4.2.22 HAL_ADC_ConvHalfCpltCallback

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

4.2.23 HAL_ADC_LevelOutOfWindowCallback

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

4.2.24 HAL_ADC_ErrorCallback

Function Name	void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)
---------------	--

	Function Description	ADC error callback in non blocking mode (ADC conversion with interruption or transfer by DMA)
	Parameters	<ul style="list-style-type: none"> hadc: ADC handle
	Return values	<ul style="list-style-type: none"> None
4.2.25	HAL_ADC_ConfigChannel	
	Function Name	HAL_StatusTypeDef HAL_ADC_ConfigChannel(ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)
	Function Description	Configures the the selected channel to be linked to the regular group.
	Parameters	<ul style="list-style-type: none"> hadc: ADC handle sConfig: Structure of ADC channel for regular group.
	Return values	<ul style="list-style-type: none"> HAL status
	Notes	<ul style="list-style-type: none"> 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). These internal paths can be disabled using function HAL_ADC_DelInit(). 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	HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig(ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)
	Function Description	Configures the analog watchdog.
	Parameters	<ul style="list-style-type: none"> hadc: ADC handle AnalogWDGConfig: Structure of ADC analog watchdog configuration
	Return values	<ul style="list-style-type: none"> HAL status
	Notes	<ul style="list-style-type: none"> 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".
4.2.27	HAL_ADC_GetState	

Function Name	<code>uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)</code>
Function Description	return the ADC state
Parameters	<ul style="list-style-type: none">• <code>hadc</code>: ADC handle
Return values	<ul style="list-style-type: none">• HAL state

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">• <code>hadc</code>: ADC handle
Return values	<ul style="list-style-type: none">• ADC Error Code

4.3 ADC Firmware driver defines

4.3.1 ADC

ADC analog watchdog mode

`ADC_ANALOGWATCHDOG_NONE`
`ADC_ANALOGWATCHDOG_SINGLE_REG`
`ADC_ANALOGWATCHDOG_ALL_REG`

ADC channels

`ADC_CHANNEL_0`
`ADC_CHANNEL_1`
`ADC_CHANNEL_2`
`ADC_CHANNEL_3`
`ADC_CHANNEL_4`
`ADC_CHANNEL_5`
`ADC_CHANNEL_6`
`ADC_CHANNEL_7`
`ADC_CHANNEL_8`
`ADC_CHANNEL_9`
`ADC_CHANNEL_10`
`ADC_CHANNEL_11`
`ADC_CHANNEL_12`
`ADC_CHANNEL_13`
`ADC_CHANNEL_14`
`ADC_CHANNEL_15`
`ADC_CHANNEL_16`
`ADC_CHANNEL_17`

ADC_CHANNEL_TEMPSENSOR

ADC_CHANNEL_VREFINT

ADC_CHANNEL_18

ADC_CHANNEL_VBAT

ADC ClockPrescaler

ADC_CLOCK_ASYNC_DIV1 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 Data_align

ADC_DATAALIGN_RIGHT

ADC_DATAALIGN_LEFT

ADC EOCSelection

ADC_EOC_SINGLE_CONV

ADC_EOC_SEQ_CONV

ADC_EOC_SINGLE_SEQ_CONV reserved for future use

ADC Error Code

HAL_ADC_ERROR_NONE No error

HAL_ADC_ERROR_INTERNAL ADC IP internal error: if problem of clocking, enable/disable, erroneous state

HAL_ADC_ERROR_OVR Overrun error

HAL_ADC_ERROR_DMA DMA transfer error

ADC Event type

ADC_AWD_EVENT ADC Analog watchdog 1 event

ADC_OVR_EVENT ADC overrun event

ADC Exported Constants

ADC_CCR_ALL

ADC Exported Macros

__HAL_ADC_ENABLE

Description:

- Enable the ADC peripheral.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_DISABLE

Description:

- Disable the ADC peripheral.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

- None

`__HAL_ADC_ENABLE_IT`**Description:**

- Enable the ADC end of conversion interrupt.

Parameters:

- `__HANDLE__`: ADC handle
- `__INTERRUPT__`: ADC Interrupt This parameter can be any combination of the following values:
 - `ADC_IT_EOC`: ADC End of Regular Conversion interrupt source
 - `ADC_IT_EOS`: ADC End of Regular sequence of Conversions interrupt source
 - `ADC_IT_AWD`: ADC Analog watchdog interrupt source
 - `ADC_IT_OVR`: ADC overrun interrupt source
 - `ADC_IT_EOSMP`: ADC End of Sampling interrupt source
 - `ADC_IT_RDY`: ADC Ready interrupt source

Return value:

- None

`__HAL_ADC_DISABLE_IT`**Description:**

- Disable the ADC end of conversion interrupt.

Parameters:

- `__HANDLE__`: ADC handle
- `__INTERRUPT__`: ADC Interrupt This parameter can be any combination of the following values:
 - `ADC_IT_EOC`: ADC End of Regular Conversion interrupt source
 - `ADC_IT_EOS`: ADC End of Regular sequence of Conversions interrupt source
 - `ADC_IT_AWD`: ADC Analog watchdog interrupt source
 - `ADC_IT_OVR`: ADC overrun interrupt source
 - `ADC_IT_EOSMP`: ADC End of Sampling interrupt source
 - `ADC_IT_RDY`: ADC Ready interrupt source

source

Return value:

- None

`_HAL_ADC_GET_IT_SOURCE`

Description:

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

Parameters:

- `_HANDLE_`: ADC handle
- `_INTERRUPT_`: ADC interrupt source to check This parameter can be any combination of the following values:
 - `ADC_IT_EOC`: ADC End of Regular Conversion interrupt source
 - `ADC_IT_EOS`: ADC End of Regular sequence of Conversions interrupt source
 - `ADC_IT_AWD`: ADC Analog watchdog interrupt source
 - `ADC_IT_OVR`: ADC overrun interrupt source
 - `ADC_IT_EOSMP`: ADC End of Sampling interrupt source
 - `ADC_IT_RDY`: ADC Ready interrupt source

Return value:

- State: of interruption (SET or RESET)

`_HAL_ADC_GET_FLAG`

Description:

- Get the selected ADC's flag status.

Parameters:

- `_HANDLE_`: ADC handle
- `_FLAG_`: ADC flag This parameter can be any combination of the following values:
 - `ADC_FLAG_EOC`: ADC End of Regular conversion flag
 - `ADC_FLAG_EOS`: ADC End of Regular sequence of Conversions flag
 - `ADC_FLAG_AWD`: ADC Analog watchdog flag
 - `ADC_FLAG_OVR`: ADC overrun flag
 - `ADC_FLAG_EOSMP`: ADC End of Sampling flag
 - `ADC_FLAG_RDY`: ADC Ready flag

Return value:

- None

`_HAL_ADC_CLEAR_FLAG`

Description:

- Clear the ADC's pending flags.

Parameters:

- `__HANDLE__`: ADC handle
- `__FLAG__`: ADC flag This parameter can be any combination of the following values:
 - `ADC_FLAG_EOC`: ADC End of Regular conversion flag
 - `ADC_FLAG_EOS`: ADC End of Regular sequence of Conversions flag
 - `ADC_FLAG_AWD`: ADC Analog watchdog flag
 - `ADC_FLAG_OVR`: ADC overrun flag
 - `ADC_FLAG_EOSMP`: ADC End of Sampling flag
 - `ADC_FLAG_RDY`: ADC Ready flag

Return value:

- None

`__HAL_ADC_RESET_HANDLE_STATE`

Description:

- Reset ADC handle state.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

- None

ADC Exported Types

<code>HAL_ADC_STATE_RESET</code>	ADC not yet initialized or disabled
<code>HAL_ADC_STATE_READY</code>	ADC peripheral ready for use
<code>HAL_ADC_STATE_BUSY_INTERNAL</code>	ADC is busy to internal process (initialization, calibration)
<code>HAL_ADC_STATE_TIMEOUT</code>	TimeOut occurrence
<code>HAL_ADC_STATE_ERROR_INTERNAL</code>	Internal error occurrence
<code>HAL_ADC_STATE_ERROR_CONFIG</code>	Configuration error occurrence
<code>HAL_ADC_STATE_ERROR_DMA</code>	DMA error occurrence
<code>HAL_ADC_STATE_REG_BUSY</code>	A conversion on group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on, multimode ADC master control)
<code>HAL_ADC_STATE_REG_EOC</code>	Conversion data available on group regular
<code>HAL_ADC_STATE_REG_OVR</code>	Overrun occurrence
<code>HAL_ADC_STATE_REG_EOSMP</code>	Not available on STM32F0 device: End Of Sampling flag raised
<code>HAL_ADC_STATE_INJ_BUSY</code>	Not available on STM32F0 device: A conversion on group injected is ongoing or can occur (either by auto-injection mode,

	external trigger, low power auto power-on, multimode ADC master control)
HAL_ADC_STATE_INJ_EOC	Not available on STM32F0 device: Conversion data available on group injected
HAL_ADC_STATE_INJ_JQOVF	Not available on STM32F0 device: Not available on STM32F0 device: Injected queue overflow occurrence
HAL_ADC_STATE_AWD1	Out-of-window occurrence of analog watchdog 1
HAL_ADC_STATE_AWD2	Not available on STM32F0 device: Out-of-window occurrence of analog watchdog 2
HAL_ADC_STATE_AWD3	Not available on STM32F0 device: Out-of-window occurrence of analog watchdog 3
HAL_ADC_STATE_MULTIMODE_SLAVE	Not available on STM32F0 device: ADC in multimode slave state, controlled by another ADC master (

ADC External trigger edge Regular

ADC_EXTERNALTRIGCONVEDGE_NONE
 ADC_EXTERNALTRIGCONVEDGE_RISING
 ADC_EXTERNALTRIGCONVEDGE_FALLING
 ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

ADC External trigger source Regular

ADC_EXTERNALTRIGCONV_T1_TRGO
 ADC_EXTERNALTRIGCONV_T1_CC4
 ADC_EXTERNALTRIGCONV_T3_TRGO
 ADC_SOFTWARE_START
 ADC_EXTERNALTRIGCONV_T2_TRGO
 ADC_EXTERNALTRIGCONV_T15_TRGO

ADC flags definition

ADC_FLAG_AWD	ADC Analog watchdog flag
ADC_FLAG_OVR	ADC overrun flag
ADC_FLAG_EOS	ADC End of Regular sequence of Conversions flag
ADC_FLAG_EOC	ADC End of Regular Conversion flag
ADC_FLAG_EOSMP	ADC End of Sampling flag
ADC_FLAG_RDY	ADC Ready flag

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

ADC_OVR_DATA_OVERWRITTEN
ADC_OVR_DATA_PRESERVED

ADC range verification

IS_ADC_RANGE

ADC rank

ADC_RANK_CHANNEL_NUMBER	Enable the rank of the selected channels. Number of ranks in the sequence is defined by number of channels enabled, rank of each channel is defined by channel number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...)
ADC_RANK_NONE	Disable the selected rank (selected channel) from sequencer

ADC regular rank verification

IS_ADC_REGULAR_RANK

ADC Resolution

ADC_RESOLUTION_12B	ADC 12-bit resolution
ADC_RESOLUTION_10B	ADC 10-bit resolution
ADC_RESOLUTION_8B	ADC 8-bit resolution
ADC_RESOLUTION_6B	ADC 6-bit 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

ADC Scan mode

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	

5 HAL ADC Extension Driver

5.1 ADCEx Firmware driver API description

5.1.1 IO operation functions

This section provides functions allowing to:

- Perform the ADC calibration.

This section contains the following APIs:

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

5.1.2 HAL_ADCEx_Calibration_Start

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

6 HAL CAN Generic Driver

6.1 CAN Firmware driver registers structures

6.1.1 CAN_InitTypeDef

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

Data Fields

- ***uint32_t FilterIdHigh***
- ***uint32_t FilterIdLow***
- ***uint32_t FilterMaskIdHigh***
- ***uint32_t FilterMaskIdLow***
- ***uint32_t FilterFIFOAssignment***
- ***uint32_t FilterNumber***
- ***uint32_t FilterMode***
- ***uint32_t FilterScale***
- ***uint32_t FilterActivation***
- ***uint32_t BankNumber***

Field Documentation

- ***uint32_t CAN_FilterConfTypeDef::FilterIdHigh***
Specifies the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t CAN_FilterConfTypeDef::FilterIdLow***
Specifies the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t CAN_FilterConfTypeDef::FilterMaskIdHigh***
Specifies the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t CAN_FilterConfTypeDef::FilterMaskIdLow***
Specifies the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t CAN_FilterConfTypeDef::FilterFIFOAssignment***
Specifies the FIFO (0 or 1) which will be assigned to the filter. This parameter can be a value of [CAN_filter_FIFO](#)
- ***uint32_t CAN_FilterConfTypeDef::FilterNumber***
Specifies the filter which will be initialized. This parameter must be a number between Min_Data = 0 and Max_Data = 27.
- ***uint32_t CAN_FilterConfTypeDef::FilterMode***
Specifies the filter mode to be initialized. This parameter can be a value of [CAN_filter_mode](#)
- ***uint32_t CAN_FilterConfTypeDef::FilterScale***
Specifies the filter scale. This parameter can be a value of [CAN_filter_scale](#)

- ***uint32_t CAN_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

Data Fields

- ***uint32_t StdId***
- ***uint32_t ExtId***
- ***uint32_t IDE***
- ***uint32_t RTR***
- ***uint32_t DLC***
- ***uint8_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.
- ***uint8_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

Data Fields

- ***uint32_t StdId***
- ***uint32_t ExtId***
- ***uint32_t IDE***
- ***uint32_t RTR***
- ***uint32_t DLC***
- ***uint8_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.
- ***uint8_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

Data Fields

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

Field Documentation

- ***CAN_TypeDef* CAN_HandleTypeDef::Instance***
Register base address
- ***CAN_InitTypeDef CAN_HandleTypeDef::Init***
CAN required parameters
- ***CanTxMsgTypeDef* CAN_HandleTypeDef::pTxMsg***
Pointer to transmit structure
- ***CanRxMsgTypeDef* CAN_HandleTypeDef::pRxMsg***
Pointer to reception structure

- **`HAL_LockTypeDef CAN_HandleTypeDef::Lock`**
CAN locking object
- **`_IO HAL_CAN_StateTypeDef CAN_HandleTypeDef::State`**
CAN communication state
- **`_IO uint32_t CAN_HandleTypeDef::ErrorCode`**
CAN Error code This parameter can be a value of `HAL_CAN_Error_Code`

6.2 CAN Firmware driver API description

6.2.1 How to use this driver

1. Enable the CAN controller interface clock using
`_HAL_RCC_CAN1_CLK_ENABLE();`
2. CAN pins configuration
 - Enable the clock for the CAN GPIOs using the following function:
`_HAL_RCC_GPIOx_CLK_ENABLE();`
 - Connect and configure the involved CAN pins 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.

This section contains the following APIs:

- `HAL_CAN_Init\(\)`
- `HAL_CAN_ConfigFilter\(\)`
- `HAL_CAN_DelInit\(\)`
- `HAL_CAN_MspInit\(\)`
- `HAL_CAN_MspDelInit\(\)`

6.2.3 Peripheral State and Error functions

This subsection provides functions allowing to :

- Check the CAN state.
- Check CAN Errors detected during interrupt process

This section contains the following APIs:

- `HAL_CAN_GetState\(\)`
- `HAL_CAN_GetError\(\)`

6.2.4 HAL_CAN_Init

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

6.2.5 HAL_CAN_ConfigFilter

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

that contains the filter configuration information.

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

6.2.6 HAL_CAN_DeInit

Function Name	HAL_StatusTypeDef HAL_CAN_DeInit (CAN_HandleTypeDef * hcan)
Function Description	Deinitializes the CANx peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	<ul style="list-style-type: none"> HAL status

6.2.7 HAL_CAN_MspInit

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

6.2.8 HAL_CAN_MspDeInit

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

6.2.9 HAL_CAN_Transmit

Function Name	HAL_StatusTypeDef HAL_CAN_Transmit (CAN_HandleTypeDef * hcan, uint32_t Timeout)
Function Description	Initiates and transmits a CAN frame message.
Parameters	<ul style="list-style-type: none"> hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. Timeout: Timeout duration.
Return values	<ul style="list-style-type: none"> HAL status

6.2.10 HAL_CAN_Transmit_IT

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

- Return values
- HAL status

6.2.11 HAL_CAN_Receive

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

6.2.12 HAL_CAN_Receive_IT

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

6.2.13 HAL_CAN_Sleep

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

6.2.14 HAL_CAN_WakeUp

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

6.2.15 HAL_CAN_IRQHandler

Function Name	void HAL_CAN_IRQHandler (CAN_HandleTypeDef * hcan)
---------------	---

	Function Description	Handles CAN interrupt request.
	Parameters	<ul style="list-style-type: none"> • hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
	Return values	<ul style="list-style-type: none"> • None
6.2.16 HAL_CAN_TxCpltCallback		
	Function Name	void HAL_CAN_TxCpltCallback (CAN_HandleTypeDef * hcan)
	Function Description	Transmission complete callback in non blocking mode.
	Parameters	<ul style="list-style-type: none"> • hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
	Return values	<ul style="list-style-type: none"> • None
6.2.17 HAL_CAN_RxCpltCallback		
	Function Name	void HAL_CAN_RxCpltCallback (CAN_HandleTypeDef * hcan)
	Function Description	Transmission complete callback in non blocking mode.
	Parameters	<ul style="list-style-type: none"> • hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
	Return values	<ul style="list-style-type: none"> • None
6.2.18 HAL_CAN_ErrorCallback		
	Function Name	void HAL_CAN_ErrorCallback (CAN_HandleTypeDef * hcan)
	Function Description	Error CAN callback.
	Parameters	<ul style="list-style-type: none"> • hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
	Return values	<ul style="list-style-type: none"> • None
6.2.19 HAL_CAN_GetState		
	Function Name	HAL_CAN_StateTypeDef HAL_CAN_GetState (CAN_HandleTypeDef * hcan)
	Function Description	return the CAN state
	Parameters	<ul style="list-style-type: none"> • hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
	Return values	<ul style="list-style-type: none"> • HAL state
6.2.20 HAL_CAN_GetError		
	Function Name	uint32_t HAL_CAN_GetError (CAN_HandleTypeDef * hcan)
	Function Description	Return the CAN error code.
	Parameters	<ul style="list-style-type: none"> • hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
	Return values	<ul style="list-style-type: none"> • CAN Error Code

6.3 CAN Firmware driver defines

6.3.1 CAN

CAN Exported Macros

<code>__HAL_CAN_RESET_HANDLE_STATE</code>	Description: <ul style="list-style-type: none">• Reset CAN handle state. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: CAN handle. Return value: <ul style="list-style-type: none">• None
<code>__HAL_CAN_ENABLE_IT</code>	Description: <ul style="list-style-type: none">• Enable the specified CAN interrupts. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: CAN handle.• <code>__INTERRUPT__</code>: CAN Interrupt Return value: <ul style="list-style-type: none">• None
<code>__HAL_CAN_DISABLE_IT</code>	Description: <ul style="list-style-type: none">• Disable the specified CAN interrupts. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: CAN handle.• <code>__INTERRUPT__</code>: CAN Interrupt Return value: <ul style="list-style-type: none">• None
<code>__HAL_CAN_MSG_PENDING</code>	Description: <ul style="list-style-type: none">• Return the number of pending received messages. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: CAN handle.• <code>__FIFONUMBER__</code>: Receive FIFO number, CAN_FIFO0 or CAN_FIFO1. Return value: <ul style="list-style-type: none">• The: number of pending message.
<code>__HAL_CAN_GET_FLAG</code>	Description: <ul style="list-style-type: none">• Check whether the specified CAN flag is set or not. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: specifies the CAN Handle.• <code>__FLAG__</code>: 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_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

- 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

[__HAL_CAN_TRANSMIT_STATUS](#)**Return value:**

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

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](#)**Description:**

- Cancel a transmit request.

Parameters:

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

Return value:

- None

[__HAL_CAN_DBG_FREEZE](#)**Description:**

- Enable or disables the DBG Freeze for CAN.

Parameters:

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

Return value:

- None

CAN filter FIFO

CAN_FILTER_FIFO0 Filter FIFO 0 assignment for filter x

CAN_FILTER_FIFO1 Filter FIFO 1 assignment for filter x

CAN filter mode

CAN_FILTERMODE_IDMASK Identifier mask mode

CAN_FILTERMODE_IDLIST Identifier list mode

CAN filter scale

CAN_FILTERSCALE_16BIT Two 16-bit filters

CAN_FILTERSCALE_32BIT One 32-bit filter

CAN flags

CAN_FLAG_RQCP0 Request MailBox0 flag

CAN_FLAG_RQCP1 Request MailBox1 flag

CAN_FLAG_RQCP2 Request MailBox2 flag

CAN_FLAG_TXOK0 Transmission OK MailBox0 flag

CAN_FLAG_TXOK1 Transmission OK MailBox1 flag

CAN_FLAG_TXOK2 Transmission OK MailBox2 flag

CAN_FLAG_TME0 Transmit mailbox 0 empty flag

CAN_FLAG_TME1 Transmit mailbox 0 empty flag

CAN_FLAG_TME2 Transmit mailbox 0 empty flag

CAN_FLAG_FF0 FIFO 0 Full flag

CAN_FLAG_FOV0 FIFO 0 Overrun flag

CAN_FLAG_FF1 FIFO 1 Full flag

CAN_FLAG_FOV1 FIFO 1 Overrun flag

CAN_FLAG_WKU Wake up flag

CAN_FLAG_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

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 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

CAN receive FIFO number constants

CAN_FIFO0	CAN FIFO 0 used to receive
CAN_FIFO1	CAN FIFO 1 used to receive

CAN remote transmission request

CAN_RTR_DATA	Data frame
CAN_RTR_REMOTE	Remote frame

CAN 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

CAN time quantum in bit segment 1

CAN_BS1_1TQ	1 time quantum
-------------	----------------

CAN_BS1_2TQ	2 time quantum
CAN_BS1_3TQ	3 time quantum
CAN_BS1_4TQ	4 time quantum
CAN_BS1_5TQ	5 time quantum
CAN_BS1_6TQ	6 time quantum
CAN_BS1_7TQ	7 time quantum
CAN_BS1_8TQ	8 time quantum
CAN_BS1_9TQ	9 time quantum
CAN_BS1_10TQ	10 time quantum
CAN_BS1_11TQ	11 time quantum
CAN_BS1_12TQ	12 time quantum
CAN_BS1_13TQ	13 time quantum
CAN_BS1_14TQ	14 time quantum
CAN_BS1_15TQ	15 time quantum
CAN_BS1_16TQ	16 time quantum

CAN time quantum in bit segment 2

CAN_BS2_1TQ	1 time quantum
CAN_BS2_2TQ	2 time quantum
CAN_BS2_3TQ	3 time quantum
CAN_BS2_4TQ	4 time quantum
CAN_BS2_5TQ	5 time quantum
CAN_BS2_6TQ	6 time quantum
CAN_BS2_7TQ	7 time quantum
CAN_BS2_8TQ	8 time quantum

7 HAL CEC Generic Driver

7.1 CEC Firmware driver registers structures

7.1.1 CEC_InitTypeDef

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

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

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 customed 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)
 - BREErErrorBitGen (Error-Bit generation in case of Bit Rising Error)
 - LBPErErrorBitGen (Error-Bit generation in case of Long Bit Period Error)
 - BroadcastMsgNoErrorBitGen (Error-bit generation in case of broadcast message error)
 - SignalFreeTimeOption (SFT Timer start definition)
 - OwnAddress (CEC device address)
 - ListenMode

This section contains the following APIs:

- [**HAL_CEC_Init\(\)**](#)
- [**HAL_CEC_DelInit\(\)**](#)
- [**HAL_CEC_MspInit\(\)**](#)
- [**HAL_CEC_MspDelInit\(\)**](#)

7.2.3 IO operation function

This section contains the following APIs:

- [**HAL_CEC_Transmit\(\)**](#)
- [**HAL_CEC_Receive\(\)**](#)
- [**HAL_CEC_Transmit_IT\(\)**](#)
- [**HAL_CEC_Receive_IT\(\)**](#)

- [*HAL_CEC_GetReceivedFrameSize\(\)*](#)
- [*HAL_CEC_IRQHandler\(\)*](#)
- [*HAL_CEC_TxCpltCallback\(\)*](#)
- [*HAL_CEC_RxCpltCallback\(\)*](#)
- [*HAL_CEC_ErrorCallback\(\)*](#)

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.

This section contains the following APIs:

- [*HAL_CEC_GetState\(\)*](#)
- [*HAL_CEC_GetError\(\)*](#)

7.2.5 HAL_CEC_Init

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

7.2.6 HAL_CEC_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_CEC_DelInit (CEC_HandleTypeDef * hcec)</code>
Function Description	Delinitializes the CEC peripheral.
Parameters	<ul style="list-style-type: none"> • hcec: CEC handle
Return values	<ul style="list-style-type: none"> • HAL status

7.2.7 HAL_CEC_MspInit

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

7.2.8 HAL_CEC_MspDelInit

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

7.2.9 HAL_CEC_Transmit

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

7.2.10 HAL_CEC_Receive

Function Name	HAL_StatusTypeDef HAL_CEC_Receive (CEC_HandleTypeDef * hcec, uint8_t * pData, uint32_t Timeout)
Function Description	Receive data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hcec: CEC handle • pData: pointer to received data buffer. • Timeout: 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 hcec->RxXferSize. hcec->RxXferSize is the sum of opcodes + operands (0 to 14 operands max). If only a header is received, hcec->RxXferSize = 0
Return values	<ul style="list-style-type: none"> • HAL status

7.2.11 HAL_CEC_Transmit_IT

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

7.2.12 HAL_CEC_Receive_IT

Function Name	HAL_StatusTypeDef HAL_CEC_Receive_IT (CEC_HandleTypeDef * hcec, uint8_t * pData)
Function Description	Receive data in interrupt mode.

Parameters	<ul style="list-style-type: none"> hcec: CEC handle pData: 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->RxXferSize. hcec->RxXferSize is the sum of opcodes + operands (0 to 14 operands max). If only a header is received, hcec->RxXferSize = 0
Return values	<ul style="list-style-type: none"> HAL status

7.2.13 HAL_CEC_GetReceivedFrameSize

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

7.2.14 HAL_CEC_IRQHandler

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

7.2.15 HAL_CEC_TxCpltCallback

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

7.2.16 HAL_CEC_RxCpltCallback

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

7.2.17 HAL_CEC_ErrorCallback

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

7.2.18 HAL_CEC_GetState

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

7.2.19 HAL_CEC_GetError

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

7.3 CEC Firmware driver defines

7.3.1 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

Reception Stop on Error

CEC_NO_RX_STOP_ON_BRE

CEC_RX_STOP_ON_BRE

Error Bit Generation on Broadcast message

CEC_BROADCASTERROR_ERRORBIT_GENERATION

CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION

CEC Error Code

HAL_CEC_ERROR_NONE no error

HAL_CEC_ERROR_RXOVR CEC Rx-Overrun

HAL_CEC_ERROR_BRE CEC Rx Bit Rising Error

HAL_CEC_ERROR_SBPE CEC Rx Short Bit period Error

HAL_CEC_ERROR_LBPE CEC Rx Long Bit period Error

HAL_CEC_ERROR_RXACKE CEC Rx Missing Acknowledge

HAL_CEC_ERROR_ARBLST CEC Arbitration Lost

HAL_CEC_ERROR_TXUDR CEC Tx-Buffer Underrun

HAL_CEC_ERROR_TXERR CEC Tx-Error

`HAL_CEC_ERROR_TXACKE` CEC Tx Missing Acknowledge

CEC Exported Macros

`_HAL_CEC_RESET_HANDLE_STATE`

Description:

- Reset CEC handle state.

Parameters:

- `_HANDLE_`: CEC handle.

Return value:

- None

`_HAL_CEC_GET_FLAG`

Description:

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

Parameters:

- `_HANDLE_`: specifies the CEC Handle.
- `_FLAG_`: specifies the interrupt to check.
 - `CEC_FLAG_TXACKE`: Tx Missing acknowledge Error
 - `CEC_FLAG_TXERR`: Tx Error.
 - `CEC_FLAG_TXUDR`: Tx-Buffer Underrun.
 - `CEC_FLAG_TXEND`: End of transmission (successful transmission of the last byte).
 - `CEC_FLAG_TXBR`: Tx-Byte Request.
 - `CEC_FLAG_ARBLST`: Arbitration Lost
 - `CEC_FLAG_RXACKE`: Rx-Missing Acknowledge
 - `CEC_FLAG_LBPE`: Rx Long period Error
 - `CEC_FLAG_SBPE`: Rx Short period Error
 - `CEC_FLAG_BRE`: Rx Bit Rissing Error
 - `CEC_FLAG_RXOVR`: Rx Overrun.
 - `CEC_FLAG_RXEND`: End Of Reception.
 - `CEC_FLAG_RXBR`: Rx-Byte Received.

Return value:

- None

[__HAL_CEC_CLEAR_FLAG](#)**Description:**

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

Parameters:

- [__HANDLE__](#): specifies the CEC Handle.
- [__FLAG__](#): specifies the interrupt/status flag to clear. This parameter can be one of the following values:
 - [CEC_FLAG_TXACKE](#): Tx Missing acknowledge Error
 - [CEC_FLAG_TXERR](#): Tx Error.
 - [CEC_FLAG_TXUDR](#): Tx Buffer Underrun.
 - [CEC_FLAG_TXEND](#): End of transmission (successful transmission of the last byte).
 - [CEC_FLAG_TXBR](#): Tx Byte Request.
 - [CEC_FLAG_ARBLST](#): Arbitration Lost
 - [CEC_FLAG_RXACKE](#): Rx-Missing Acknowledge
 - [CEC_FLAG_LBPE](#): Rx Long period Error
 - [CEC_FLAG_SBPE](#): Rx Short period Error
 - [CEC_FLAG_BRE](#): Rx Bit Rissing Error
 - [CEC_FLAG_RXOVR](#): Rx Overrun.
 - [CEC_FLAG_RXEND](#): End Of Reception.
 - [CEC_FLAG_RXBR](#): Rx-Byte Received.

Return value:

- none

[__HAL_CEC_ENABLE_IT](#)**Description:**

- Enables the specified CEC interrupt.

Parameters:

- [__HANDLE__](#): specifies the CEC Handle.
- [__INTERRUPT__](#): specifies the CEC interrupt to enable. This parameter can be one of the following values:

- CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
- CEC_IT_TXERR: Tx Error IT Enable
- CEC_IT_RXUDR: Tx- Buffer Underrun IT Enable
- CEC_IT_TXEND: End of transmission IT Enable
- CEC_IT_TXBR: Tx-Byte Request IT Enable
- CEC_IT_ARBLST: Arbitration Lost IT Enable
- CEC_IT_RXACKE: Rx- Missing Acknowledge IT Enable
- CEC_IT_LBPE: Rx Long period Error IT Enable
- CEC_IT_SBPE: Rx Short period Error IT Enable
- CEC_IT_BRE: Rx Bit Rising Error IT Enable
- CEC_IT_RXOVR: Rx Overrun IT Enable
- CEC_IT_RXEND: End Of Reception IT Enable
- CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- none

__HAL_CEC_DISABLE_IT

- Disables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to disable. This parameter can be one of the following values:
 - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
 - CEC_IT_TXERR: Tx Error IT Enable
 - CEC_IT_RXUDR: Tx- Buffer Underrun IT Enable
 - CEC_IT_TXEND: End of transmission IT Enable
 - CEC_IT_RXBR: Tx-Byte Request IT Enable

- CEC_IT_ARBLST: Arbitration Lost IT Enable
- CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
- CEC_IT_LBPE: Rx Long period Error IT Enable
- CEC_IT_SBPE: Rx Short period Error IT Enable
- CEC_IT_BRE: Rx Bit Rising Error IT Enable
- CEC_IT_RXOVR: Rx Overrun IT Enable
- CEC_IT_RXEND: End Of Reception IT Enable
- CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- none

`__HAL_CEC_GET_IT_SOURCE`

Description:

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

Parameters:

- `__HANDLE__`: specifies the CEC Handle.
- `__INTERRUPT__`: specifies the CEC interrupt to check. This parameter can be one of the following values:
 - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
 - CEC_IT_TXERR: Tx Error IT Enable
 - CEC_IT_TXUDR: Tx-Buffer Underrun IT Enable
 - CEC_IT_TXEND: End of transmission IT Enable
 - CEC_IT_TXBR: Tx-Byte Request IT Enable
 - CEC_IT_ARBLST: Arbitration Lost IT Enable
 - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
 - CEC_IT_LBPE: Rx Long period Error IT Enable
 - CEC_IT_SBPE: Rx Short period Error IT Enable
 - CEC_IT_BRE: Rx Bit Rising Error IT Enable

- Rising Error IT Enable
- CEC_IT_RXOVR: Rx Overrun IT Enable
- CEC_IT_RXEND: End Of Reception IT Enable
- CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- FlagStatus

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.

`__HAL_CEC_GET_TRANSMISSION_START_FLAG`

Description:

- Get Transmission Start flag.

Parameters:

- `__HANDLE__`: specifies the CEC Handle.

Return value:

- `FlagStatus`

`__HAL_CEC_GET_TRANSMISSION_END_FLAG`

Description:

- Get Transmission End flag.

Parameters:

- `__HANDLE__`: specifies the CEC Handle.

Return value:

- `FlagStatus`

`__HAL_CEC_CLEAR_OAR`

Description:

- Clear OAR register.

Parameters:

- `__HANDLE__`: specifies the CEC Handle.

Return value:

- none

`__HAL_CEC_SET_OAR`

Description:

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

Parameters:

- `__HANDLE__`: specifies the CEC Handle.
- `__ADDRESS__`: Own Address value (CEC logical address is identified by bit position)

Return value:

- none

CEC Flags definition

`CEC_FLAG_TXACKE`

`CEC_FLAG_TXERR`

`CEC_FLAG_TXUDR`

`CEC_FLAG_TXEND`

CEC_FLAG_TXBR
CEC_FLAG_ARBLST
CEC_FLAG_RXACKE
CEC_FLAG_LBPE
CEC_FLAG_SBPE
CEC_FLAG_BRE
CEC_FLAG_RXOVR
CEC_FLAG_RXEND
CEC_FLAG_RXBR

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

CEC Interrupts definition

CEC_IT_TXACKE
CEC_IT_TXERR
CEC_IT_TXUDR
CEC_IT_TXEND
CEC_IT_TXBR
CEC_IT_ARBLST
CEC_IT_RXACKE
CEC_IT_LBPE
CEC_IT_SBPE
CEC_IT_BRE
CEC_IT_RXOVR
CEC_IT_RXEND
CEC_IT_RXBR

Error Bit Generation if Long Bit Period Error reported

CEC_LBPE_ERRORBIT_NO_GENERATION
CEC_LBPE_ERRORBIT_GENERATION

Listening mode option

CEC_REDUCED_LISTENING_MODE
CEC_FULL_LISTENING_MODE

Device Own Address position in CEC CFGR register

CEC_CFGR_OAR_LSB_POS

Signal Free Time start option

CEC_SFT_START_ON_TXSOM

CEC_SFT_START_ON_TX_RX_END

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

Receiver Tolerance

CEC_STANDARD_TOLERANCE

CEC_EXTENDED_TOLERANCE

8 HAL COMP Generic Driver

8.1 COMP Firmware driver registers structures

8.1.1 COMP_InitTypeDef

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

Data Fields

- ***COMP_TypeDef * Instance***
- ***COMP_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t 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 uint32_t COMP_HandleTypeDef::State***
COMP communication state This parameter can be a value of ***COMP_State***

8.2 COMP Firmware driver API description

8.2.1 COMP Peripheral features

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

- The non inverting input and inverting input can be set to GPIO pins as shown in [Table 16: "COMP Inputs for STM32F05xx, STM32F07x and STM32F09x devices"](#).
- The COMP output is available using HAL_COMP_GetOutputLevel() and can be set on GPIO pins. Refer to [Table 17: "COMP outputs for STM32F05xx, STM32F07x and STM32F09x devices"](#).
- The COMP output can be redirected to embedded timers (TIM1, TIM2 and TIM3) Refer to [Table 18: "Redirection of COMP outputs to embedded timers for STM32F05xx, STM32F07x and STM32F09x devices"](#).
- The comparators COMP1 and COMP2 can be combined in window mode.
- 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 macros __HAL_COMP_COMP1_EXTI_GET_FLAG() and __HAL_COMP_COMP2_EXTI_GET_FLAG().

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 (input minus)
 - Select the non inverting input (input plus)
 - Select the output polarity
 - Select the output redirection
 - Select the hysteresis level
 - Select the power mode
 - Select the event/interrupt mode
 - Select the window mode HAL_COMP_Init() calls internally __HAL_RCC_SYSCFG_CLK_ENABLE() in order to access the comparator(s) registers.
3. Enable the comparator using HAL_COMP_Start() function or HAL_COMP_Start_IT() function for interrupt mode.
4. Use HAL_COMP_TriggerCallback() and/or HAL_COMP_GetOutputLevel() functions to manage comparator outputs (event/interrupt triggered and output level).
5. Disable the comparator using HAL_COMP_Stop() or HAL_COMP_Stop_IT() function.
6. De-initialize the comparator using HAL_COMP_DelInit() function.
7. For safety purposes comparator(s) can be locked using HAL_COMP_Lock() function. Only a MCU reset can reset that protection.

8.2.3 Initialization and Configuration functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:

- [*HAL_COMP_Init\(\)*](#)
- [*HAL_COMP_DelInit\(\)*](#)
- [*HAL_COMP_MspInit\(\)*](#)
- [*HAL_COMP_MspDeInit\(\)*](#)

8.2.4 IO operation functions

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

This section contains the following APIs:

- [*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.

This section contains the following APIs:

- [*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.

This section contains the following APIs:

- [*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"> • hcomp: COMP handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

8.2.8 HAL_COMP_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_COMP_DelInit (COMP_HandleTypeDef * hcomp)</code>
---------------	--

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

8.2.9 HAL_COMP_MsplInit

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

8.2.10 HAL_COMP_MspDeInit

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

8.2.11 HAL_COMP_Start

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

8.2.12 HAL_COMP_Stop

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

8.2.13 HAL_COMP_Start_IT

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

8.2.14 HAL_COMP_Stop_IT

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

8.2.15 HAL_COMP_IRQHandler

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

8.2.16 HAL_COMP_Lock

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

8.2.17 HAL_COMP_GetOutputLevel

Function Name	uint32_t HAL_COMP_GetOutputLevel (COMP_HandleTypeDef * hcomp)
Function Description	Return the output level (high or low) of the selected comparator.

8.2.18 HAL_COMP_TriggerCallback

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

8.2.19 HAL_COMP_GetState

Function Name	uint32_t HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)
Function Description	Return the COMP state.
Parameters	<ul style="list-style-type: none"> • hcomp: : COMP handle
Return values	<ul style="list-style-type: none"> • HAL state

8.3 COMP Firmware driver defines

8.3.1 COMP

COMP Exported Macros

`_HAL_COMP_RESET_HANDLE_STATE`

Description:

- Reset COMP handle state.

Parameters:

- `_HANDLE_`: COMP handle.

Return value:

- None

Description:

- Enable the specified comparator.

Parameters:

- `_HANDLE_`: COMP handle.

Return value:

- None

Description:

- Disable the specified comparator.

Parameters:

- `_HANDLE_`: COMP handle.

Return value:

- None

Description:

- Lock the specified comparator configuration.

Parameters:

- `_HANDLE_`: COMP handle.

Return value:

- None

Description:

- Enable the COMP1 EXTI line rising edge trigger.

Return value:

- None

Description:

- Disable the COMP1 EXTI line rising edge trigger.

Return value:

- None

Description:

- Enable the COMP1 EXTI line falling edge trigger.

Return value:

- None

Description:

- Disable the COMP1 EXTI line falling edge trigger.

Return value:

- None

Description:

- Enable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

Description:

- Disable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

Description:

- Enable the COMP1 EXTI line in interrupt mode.

Return value:

- None

Description:

- Disable the COMP1 EXTI line in interrupt mode.

Return value:

- None

`_HAL_COMP_COMP1_EXTI_GENERATE_SWIT`

Description:

- Generate a software interrupt on the COMP1 EXTI line.

Return value:

- None

`_HAL_COMP_COMP1_EXTI_ENABLE_EVENT`

Description:

- Enable the COMP1 EXTI Line in event mode.

Return value:

- None

`_HAL_COMP_COMP1_EXTI_DISABLE_EVENT`

Description:

- Disable the COMP1 EXTI Line in event mode.

Return value:

- None

`_HAL_COMP_COMP1_EXTI_GET_FLAG`

Description:

- Check whether the COMP1 EXTI line flag is set or not.

Return value:

- RESET: or SET

`_HAL_COMP_COMP1_EXTI_CLEAR_FLAG`

Description:

- Clear the COMP1 EXTI flag.

Return value:

- None

`_HAL_COMP_COMP2_EXTI_ENABLE_RISING_EDGE`

Description:

- Enable the COMP2 EXTI line rising edge trigger.

Return value:

- None

`_HAL_COMP_COMP2_EXTI_DISABLE_RISING_EDGE`

Description:

- Disable the COMP2 EXTI line rising edge trigger.

Return value:

- None

`_HAL_COMP_COMP2_EXTI_ENABLE_FALLING_EDGE`

Description:

- Enable the COMP2 EXTI

line falling edge trigger.

Return value:

- None

Description:

- Disable the COMP2 EXTI line falling edge trigger.

Return value:

- None

Description:

- Enable the COMP2 EXTI line rising & falling edge trigger.

Return value:

- None

Description:

- Disable the COMP2 EXTI line rising & falling edge trigger.

Return value:

- None

Description:

- Enable the COMP2 EXTI line in interrupt mode.

Return value:

- None

Description:

- Disable the COMP2 EXTI line in interrupt mode.

Return value:

- None

Description:

- Generate a software interrupt on the COMP2 EXTI line.

Return value:

- None

Description:

- Enable the COMP2 EXTI Line in event mode.

Return value:

- None

`__HAL_COMP_COMP2_EXTI_DISABLE_EVENT`**Description:**

- Disable the COMP2 EXTI Line in event mode.

Return value:

- None

`__HAL_COMP_COMP2_EXTI_GET_FLAG`**Description:**

- Check whether the COMP2 EXTI line flag is set or not.

Return value:

- RESET: or SET

`__HAL_COMP_COMP2_EXTI_CLEAR_FLAG`**Description:**

- Clear the COMP2 EXTI flag.

Return value:

- None

`__HAL_COMP_GET_FLAG`**Description:**

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

Parameters:

- `__HANDLE__`: specifies the COMP Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `COMP_FLAG_LOCK`
K: lock flag

Return value:

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

COMP EXTI Lines`COMP_EXTI_LINE_COMP1` EXTI line 21 connected to COMP1 output`COMP_EXTI_LINE_COMP2` EXTI line 22 connected to COMP2 output***COMP Flag***`COMP_FLAG_LOCK` Lock flag***COMP Private macros to get EXTI line associated with Comparators***

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 Hysteresis

COMP_HYSTERESIS_NONE	No hysteresis
COMP_HYSTERESIS_LOW	Hysteresis level low
COMP_HYSTERESIS_MEDIUM	Hysteresis level medium
COMP_HYSTERESIS_HIGH	Hysteresis level high

COMP InvertingInput

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

COMP Private macros to check input parameters

IS_COMP_OUTPUTPOL
 IS_COMP_HYSTERESIS
 IS_COMP_MODE
 IS_COMP_INVERTINGINPUT
 IS_COMP_NONINVERTINGINPUT
 IS_COMP_OUTPUT
 IS_COMP_WINDOWMODE
 IS_COMP_TRIGGERMODE

COMP Lock

COMP_LOCK_DISABLE

COMP_LOCK_ENABLE

COMP_STATE_BIT_LOCK

COMP Mode

COMP_MODE_HIGHSPEED High Speed

COMP_MODE_MEDIUMSPEED Medium Speed

COMP_MODE_LOWPOWER Low power mode

COMP_MODE_ULTRALOWPOWER Ultra-low power mode

COMP NonInvertingInput

COMP_NONINVERTINGINPUT_IO1 I/O1 (PA1 for COMP1, PA3 for COMP2) connected to comparator non inverting input

COMP_NONINVERTINGINPUT_DAC1SWITCHCLOSED DAC output connected to comparator COMP1 non inverting input

COMP Output

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

COMP OutputLevel

COMP_OUTPUTLEVEL_LOW

COMP_OUTPUTLEVEL_HIGH

COMP OutputPolarity

COMP_OUTPUTPOL_NONINVERTED COMP output on GPIO isn't inverted

COMP_OUTPUTPOL_INVERTED COMP output on GPIO is inverted

COMP State

HAL_COMP_STATE_RESET COMP not yet initialized or disabled

HAL_COMP_STATE_READY COMP initialized and ready for use

HAL_COMP_STATE_READY_LOCKED COMP initialized but the configuration is locked

HAL_COMP_STATE_BUSY COMP is running

HAL_COMP_STATE_BUSY_LOCKED COMP is running and the configuration is locked

COMP TriggerMode

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
COMP_TRIGGERMODE_EVENT_RISING	Event Mode with Rising edge trigger detection
COMP_TRIGGERMODE_EVENT_FALLING	Event Mode with Falling edge trigger detection
COMP_TRIGGERMODE_EVENT_RISING_FALLING	Event Mode with Rising/Falling edge trigger detection

COMP WindowMode

COMP_WINDOWMODE_DISABLE	Window mode disabled
COMP_WINDOWMODE_ENABLE	Window mode enabled: non inverting input of comparator 2 is connected to the non inverting input of comparator 1 (PA1)

9 HAL CORTEX Generic Driver

9.1 CORTEX Firmware driver API description

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() Negative value of IRQn_Type are not allowed.

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

This section contains the following APIs:

- `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.

This section contains the following APIs:

- `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"> • IRQn: External interrupt number . This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file) • PreemptPriority: The preemption priority for the IRQn channel. This parameter can be a value between 0 and 3. A lower priority value indicates a higher priority • SubPriority: the subpriority level for the IRQ channel. with stm32f0xx devices, this parameter is a dummy value and it is ignored, because no subpriority supported in Cortex M0 based products.
Return values	<ul style="list-style-type: none"> • None

9.1.5 HAL_NVIC_EnableIRQ

Function Name	<code>void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)</code>
Function Description	Enables a device specific interrupt in the NVIC interrupt controller.
Parameters	<ul style="list-style-type: none"> • IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f0xxxx.h))
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

9.1.6 HAL_NVIC_DisableIRQ



Function Name	void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)
Function Description	Disables a device specific interrupt in the NVIC interrupt controller.
Parameters	<ul style="list-style-type: none"> • IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f0xxxx.h))
Return values	<ul style="list-style-type: none"> • None

9.1.7 HAL_NVIC_SystemReset

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

9.1.8 HAL_SYSTICK_Config

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

9.1.9 HAL_NVIC_GetPriority

Function Name	uint32_t HAL_NVIC_GetPriority (IRQn_Type IRQn)
Function Description	Gets the priority of an interrupt.
Parameters	<ul style="list-style-type: none"> • IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f0xxxx.h))
Return values	<ul style="list-style-type: none"> • None

9.1.10 HAL_NVIC_SetPendingIRQ

Function Name	void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)
Function Description	Sets Pending bit of an external interrupt.
Parameters	<ul style="list-style-type: none"> • IRQn: External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f0xxxx.h))
Return values	<ul style="list-style-type: none"> • None

9.1.11 HAL_NVIC_GetPendingIRQ

Function Name	uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)
---------------	---

Function Description	Gets Pending Interrupt (reads the pending register in the NVIC and returns the pending bit for the specified interrupt).
Parameters	<ul style="list-style-type: none"> IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f0xxxx.h))
Return values	<ul style="list-style-type: none"> status - 0 Interrupt status is not pending. 1 Interrupt status is pending.

9.1.12 HAL_NVIC_ClearPendingIRQ

Function Name	void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)
Function Description	Clears the pending bit of an external interrupt.
Parameters	<ul style="list-style-type: none"> IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f0xxxx.h))
Return values	<ul style="list-style-type: none"> None

9.1.13 HAL_SYSTICK_CLKSourceConfig

Function Name	void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)
Function Description	Configures the SysTick clock source.
Parameters	<ul style="list-style-type: none"> CLKSource: specifies the SysTick clock source. This parameter can be one of the following values: SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source. SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.
Return values	<ul style="list-style-type: none"> None

9.1.14 HAL_SYSTICK_IRQHandler

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

9.1.15 HAL_SYSTICK_Callback

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

9.2 CORTEX Firmware driver defines

9.2.1 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 SysTick clock source

`SYSTICK_CLKSOURCE_HCLK_DIV8`

`SYSTICK_CLKSOURCE_HCLK`

10 HAL CRC Generic Driver

10.1 CRC Firmware driver registers structures

10.1.1 CRC_InitTypeDef

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_Sizes](#) 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_BYTE byte-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_DISABLE no CRC inversion, or
CRC_OUTPUTDATA_INVERSION_ENABLE CRC 0x11223344 is converted into 0x22CC4488

10.1.2 CRC_HandleTypeDef

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

10.2.1 How to use this driver

1. Enable CRC AHB clock using `__HAL_RCC_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

This section contains the following APIs:

- [*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.

This section contains the following APIs:

- [*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.

This section contains the following APIs:

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

10.2.5 HAL_CRC_Init

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

10.2.6 HAL_CRC_DeInit

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

Function Description	Deinitializes the CRC peripheral.
Parameters	<ul style="list-style-type: none">• hcrc: CRC handle
Return values	<ul style="list-style-type: none">• HAL status

10.2.7 HAL_CRC_Msplnit

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

10.2.8 HAL_CRC_MspDelnit

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

10.2.9 HAL_CRC_Accumulate

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

10.2.10 HAL_CRC_Calculate

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

10.2.11 HAL_CRC_GetState

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

10.3 CRC Firmware driver defines

10.3.1 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**Description:**

- Set CRC INIT non-default value.

Parameters:

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

Return value:

- None.

__HAL_CRC_SET_IDR**Description:**

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

Parameters:

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

Return value:

- None

__HAL_CRC_GET_IDR**Description:**

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

Parameters:

- __HANDLE__: CRC handle

Return value:

- 8-bit: value of the ID register

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

11.1.1 How to use this driver

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

11.1.2 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_CRCEx_Init straight away reports HAL_OK.
- Set the generating polynomial

This section contains the following APIs:

- [`HAL_CRCEx_Init\(\)`](#)
- [`HAL_CRCEx_Input_Data_Reverse\(\)`](#)
- [`HAL_CRCEx_Output_Data_Reverse\(\)`](#)
- [`HAL_CRCEx_Polynomial_Set\(\)`](#)

11.1.3 `HAL_CRCEx_Init`

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

11.1.4 `HAL_CRCEx_Input_Data_Reverse`

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

11.1.5 HAL_CRCEx_Output_Data_Reverse

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

11.1.6 HAL_CRCEx_Polynomial_Set

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

11.2 CRCEEx Firmware driver defines

11.2.1 CRCEEx

CRCEEx Exported Macros

`_HAL_CRC_OUTPUTREVERSAL_ENABLE`

Description:

- Set CRC output reversal.

Parameters:

- `_HANDLE_`: : CRC handle

Return value:

- None.

`_HAL_CRC_OUTPUTREVERSAL_DISABLE`

Description:

- Unset CRC output reversal.

Parameters:

- `_HANDLE_`: : CRC handle

Return value:

- None.

`_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_DISABLE`

`CRC_OUTPUTDATA_INVERSION_ENABLE`

`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

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

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

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_PIN_9) using DAC_TRIGGER_EXT_IT9. The used pin (GPIOx_PIN_9) 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.

GPIO configurations guidelines

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

DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave
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 correspondance

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

$$\text{DAC_OUTx} = \text{VREF+} * \text{DOR} / 4095$$

- with DOR is the Data Output Register

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

e.g. To set DAC_OUT1 to 0.7V, use

- Assuming that VREF+ = 3.3V, $\text{DAC_OUT1} = (3.3 * 868) / 4095 = 0.7V$

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()



For Dual mode and specific signal (Triangle and noise) generation please refer to Extended 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, use the HAL_DAC_GetValue() function.
- Stop the DAC peripheral using HAL_DAC_Stop()

DMA mode IO operation

- Start the DAC peripheral using HAL_DAC_Start_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion
- At the middle of data transfer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()

- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- 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
- In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() and 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.

This section contains the following APIs:

- `HAL_DAC_Init()`
- `HAL_DAC_DelInit()`
- `HAL_DAC_MspInit()`
- `HAL_DAC_MspDelInit()`

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.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- `HAL_DAC_Start()`
- `HAL_DAC_Stop()`

- [*HAL_DAC_Start_DMA\(\)*](#)
- [*HAL_DAC_Stop_DMA\(\)*](#)
- [*HAL_DAC_IRQHandler\(\)*](#)
- [*HAL_DAC_SetValue\(\)*](#)
- [*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.
- Get result of conversion.

This section contains the following APIs:

- [*HAL_DAC_GetValue\(\)*](#)
- [*HAL_DAC_ConfigChannel\(\)*](#)

12.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.

This section contains the following APIs:

- [*HAL_DAC_GetState\(\)*](#)
- [*HAL_DAC_GetError\(\)*](#)

12.2.7 HAL_DAC_Init

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

12.2.8 HAL_DAC_DeInit

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

12.2.9 HAL_DAC_MspInit

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

12.2.10 HAL_DAC_MspDeInit

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

12.2.11 HAL_DAC_Start

Function Name	HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)
Function Description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> • HAL status

12.2.12 HAL_DAC_Stop

Function Name	HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)
Function Description	Disables DAC and stop conversion of channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> • HAL status

12.2.13 HAL_DAC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)
Function Description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that

- contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
- **pData:** The destination peripheral Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
DAC_ALIGN_8B_R: 8bit right data alignment
selectedDAC_ALIGN_12B_L: 12bit left data alignment
selectedDAC_ALIGN_12B_R: 12bit right data alignment selected

Return values

- HAL status

12.2.14 HAL_DAC_Stop_DMA**Function Name**

HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function Description

Disables DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected

Return values

- HAL status

12.2.15 HAL_DAC_IRQHandler**Function Name**

void HAL_DAC_IRQHandler (DAC_HandleTypeDef * hdac)

Function Description

Handles DAC interrupt request.

Parameters

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

Return values

- None

12.2.16 HAL_DAC_SetValue**Function Name**

HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)

Function Description

Set the specified data holding register value for DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
- **Alignment:** Specifies the data alignment. This parameter

can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selectedDAC_ALIGN_12B_L: 12bit left data alignment selectedDAC_ALIGN_12B_R: 12bit right data alignment selected

- **Data:** Data to be loaded in the selected data holding register.
- HAL status

12.2.17 HAL_DAC_ConvCpltCallbackCh1

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

12.2.18 HAL_DAC_ConvHalfCpltCallbackCh1

Function Name	<code>void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)</code>
Function Description	Conversion half DMA transfer callback in non-blocking mode for Channel1.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • None

12.2.19 HAL_DAC_ErrorCallbackCh1

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

12.2.20 HAL_DAC_DMADebugCallbackCh1

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

12.2.21 HAL_DAC_GetValue

Function Name	<code>uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac,</code>
---------------	---

uint32_t Channel)

Function Description	Returns the last data output value of the selected DAC channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> • The selected DAC channel data output value.

12.2.22 HAL_DAC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)
Function Description	Configures the selected DAC channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • sConfig: DAC configuration structure. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> • HAL status

12.2.23 HAL_DAC_GetState

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

12.2.24 HAL_DAC_GetError

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

12.3 DAC Firmware driver defines**12.3.1 DAC*****DAC Channel selection***

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC data alignment

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE No error

HAL_DAC_ERROR_DMAUNDERRUNCH1 DAC channel1 DMA underrun error

HAL_DAC_ERROR_DMAUNDERRUNCH2 DAC channel2 DMA underrun error

HAL_DAC_ERROR_DMA DMA error

DAC Exported Macros

__HAL_DAC_RESET_HANDLE_STATE **Description:**

- Reset DAC handle state.

Parameters:

- __HANDLE__: specifies the DAC handle.

Return value:

- None

__HAL_DAC_ENABLE

Description:

- Enable the DAC channel.

Parameters:

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

Return value:

- None

__HAL_DAC_DISABLE

Description:

- Disable the DAC channel.

Parameters:

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

Return value:

- None

__HAL_DAC_ENABLE_IT

Description:

- Enable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle

- `__INTERRUPT__`: specifies the DAC interrupt. This parameter can be any combination of the following values:
 - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt
 - `DAC_IT_DMAUDR2`: DAC channel 2 DMA underrun interrupt

Return value:

- None

`__HAL_DAC_DISABLE_IT`

- Disable the DAC interrupt.

Parameters:

- `__HANDLE__`: specifies the DAC handle
- `__INTERRUPT__`: specifies the DAC interrupt. This parameter can be any combination of the following values:
 - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt

Return value:

- None

`__HAL_DAC_GET_IT_SOURCE`

- Check whether the specified DAC interrupt source is enabled or not.

Parameters:

- `__HANDLE__`: DAC handle
- `__INTERRUPT__`: DAC interrupt source to check. This parameter can be any combination of the following values:
 - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt
 - `DAC_IT_DMAUDR2`: DAC channel 2 DMA underrun interrupt

Return value:

- State: of interruption (SET or RESET)

`__HAL_DAC_GET_FLAG`

- Get the selected DAC's flag status.

Parameters:

- `__HANDLE__`: specifies the DAC handle.
- `__FLAG__`: specifies the DAC flag to get. This parameter can be any combination of the following values:
 - `DAC_FLAG_DMAUDR1`: DAC channel 1 DMA underrun flag

Return value:

- None

`_HAL_DAC_CLEAR_FLAG`

Description:

- Clear the DAC's flag.

Parameters:

- `_HANDLE_`: specifies the DAC handle.
- `_FLAG_`: specifies the DAC flag to clear. This parameter can be any combination of the following values:
 - `DAC_FLAG_DMAUDR1`: DAC channel 1 DMA underrun flag

Return value:

- None

DAC flags definition

`DAC_FLAG_DMAUDR1`

`DAC_FLAG_DMAUDR2`

DAC IT definition

`DAC_IT_DMAUDR1`

`DAC_IT_DMAUDR2`

DAC output buffer

`DAC_OUTPUTBUFFER_ENABLE`

`DAC_OUTPUTBUFFER_DISABLE`

DAC trigger selection

<code>DAC_TRIGGER_NONE</code>	Conversion is automatic once the <code>DAC1_DHRxxxx</code> register has been loaded, and not by external trigger
-------------------------------	--

<code>DAC_TRIGGER_T2_TRGO</code>	TIM2 TRGO selected as external conversion trigger for DAC channel
----------------------------------	---

<code>DAC_TRIGGER_T3_TRGO</code>	TIM3 TRGO selected as external conversion trigger for DAC channel
----------------------------------	---

<code>DAC_TRIGGER_T6_TRGO</code>	TIM6 TRGO selected as external conversion trigger for DAC channel
----------------------------------	---

<code>DAC_TRIGGER_T7_TRGO</code>	TIM7 TRGO selected as external conversion trigger for DAC channel
----------------------------------	---

<code>DAC_TRIGGER_T15_TRGO</code>	TIM15 TRGO selected as external conversion trigger for DAC channel
-----------------------------------	--

<code>DAC_TRIGGER_EXT_IT9</code>	EXTI Line9 event selected as external conversion trigger for DAC channel
----------------------------------	--

<code>DAC_TRIGGER_SOFTWARE</code>	Conversion started by software trigger for DAC channel
-----------------------------------	--

13 HAL DAC Extension Driver

13.1 DACEx Firmware driver API description

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.

This section contains the following APIs:

- [**HAL_DACEx_DualGetValue\(\)**](#)
- [**HAL_DACEx_TriangleWaveGenerate\(\)**](#)
- [**HAL_DACEx_NoiseWaveGenerate\(\)**](#)
- [**HAL_DACEx_DualSetValue\(\)**](#)
- [**HAL_DACEx_ConvCpltCallbackCh2\(\)**](#)
- [**HAL_DACEx_ConvHalfCpltCallbackCh2\(\)**](#)
- [**HAL_DACEx_ErrorCallbackCh2\(\)**](#)
- [**HAL_DACEx_DMAUnderrunCallbackCh2\(\)**](#)

13.1.3 HAL_DACEx_DualGetValue

Function Name **uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)**

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

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

Return values • The selected DAC channel data output value.

13.1.4 HAL_DACEx_TriangleWaveGenerate

Function Name **HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)**

Function Description Enables or disables the selected DAC channel wave generation.

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

13.1.5 HAL_DACEx_NoiseWaveGenerate

Function Name	HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2 • Amplitude: Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values: DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generationDAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generationDAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generationDAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generationDAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generationDAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generationDAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave

generationDAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave
 generationDAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave
 generationDAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave
 generationDAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave
 generationDAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

- Return values
- HAL status

13.1.6 HAL_DACEx_DualSetValue

Function Name	HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)
Function Description	Set the specified data holding register value for dual DAC channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Alignment: Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected • Data1: Data for DAC Channel2 to be loaded in the selected data holding register. • Data2: Data for DAC Channel1 to be loaded in the selected data holding register.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • In dual mode, a unique register access is required to write in both DAC channels at the same time.

13.1.7 HAL_DACEx_ConvCpltCallbackCh2

Function Name	void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)
Function Description	Conversion complete callback in non blocking mode for Channel2.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • None

13.1.8 HAL_DACEx_ConvHalfCpltCallbackCh2

Function Name	void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)
Function Description	Conversion half DMA transfer callback in non blocking mode for Channel2.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that

contains the configuration information for the specified DAC.

Return values • None

13.1.9 HAL_DACEx_ErrorCallbackCh2

Function Name **void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef * hdac)**

Function Description Error DAC callback for Channel2.

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

Return values • None

13.1.10 HAL_DACEx_DMAUnderrunCallbackCh2

Function Name **void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef * hdac)**

Function Description DMA underrun DAC callback for channel2.

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

Return values • None

13.1.11 HAL_DACEx_DualGetValue

Function Name **uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)**

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

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

Return values • The selected DAC channel data output value.

13.2 DACEx Firmware driver defines

13.2.1 DACEx

DACEx Ifsrnmask 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 bit[11:0] for noise wave generation
DAC_TRIANGLEAMPLITUDE_1	Select max triangle amplitude of 1
DAC_TRIANGLEAMPLITUDE_3	Select max triangle amplitude of 3
DAC_TRIANGLEAMPLITUDE_7	Select max triangle amplitude of 7
DAC_TRIANGLEAMPLITUDE_15	Select max triangle amplitude of 15
DAC_TRIANGLEAMPLITUDE_31	Select max triangle amplitude of 31
DAC_TRIANGLEAMPLITUDE_63	Select max triangle amplitude of 63
DAC_TRIANGLEAMPLITUDE_127	Select max triangle amplitude of 127
DAC_TRIANGLEAMPLITUDE_255	Select max triangle amplitude of 255
DAC_TRIANGLEAMPLITUDE_511	Select max triangle amplitude of 511
DAC_TRIANGLEAMPLITUDE_1023	Select max triangle amplitude of 1023
DAC_TRIANGLEAMPLITUDE_2047	Select max triangle amplitude of 2047
DAC_TRIANGLEAMPLITUDE_4095	Select max triangle amplitude of 4095

14 HAL DMA Generic Driver

14.1 DMA Firmware driver registers structures

14.1.1 DMA_InitTypeDef

Data Fields

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

Field Documentation

- ***uint32_t DMA_InitTypeDef::Direction***
Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of [**DMA_Data_transfer_direction**](#)
- ***uint32_t DMA_InitTypeDef::PeriphInc***
Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of [**DMA_Peripheral_incremented_mode**](#)
- ***uint32_t DMA_InitTypeDef::MemInc***
Specifies whether the memory address register should be incremented or not. This parameter can be a value of [**DMA_Memory_incremented_mode**](#)
- ***uint32_t DMA_InitTypeDef::PeriphDataAlignment***
Specifies the Peripheral data width. This parameter can be a value of [**DMA_Peripheral_data_size**](#)
- ***uint32_t DMA_InitTypeDef::MemDataAlignment***
Specifies the Memory data width. This parameter can be a value of [**DMA_Memory_data_size**](#)
- ***uint32_t DMA_InitTypeDef::Mode***
Specifies the operation mode of the DMAy Channelx. This parameter can be a value of [**DMA_mode**](#)
Note:The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel
- ***uint32_t DMA_InitTypeDef::Priority***
Specifies the software priority for the DMAy Channelx. This parameter can be a value of [**DMA_Priority_level**](#)

14.1.2 DMA_HandleTypeDef

Data Fields

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

Field Documentation

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

14.2 DMA Firmware driver API description

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.

This section contains the following APIs:

- [**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

This section contains the following APIs:

- [*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

This section contains the following APIs:

- [*HAL_DMA_GetState\(\)*](#)
- [*HAL_DMA_GetError\(\)*](#)

14.2.5 HAL_DMA_Init

Function Name	<code>HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef *hdma)</code>
Function Description	Initializes the DMA according to the specified parameters in the DMA_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hdma: Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> • HAL status

14.2.6 HAL_DMA_DeInit

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

Return values

- HAL status

14.2.7 HAL_DMA_Start

Function Name	<code>HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)</code>
---------------	---

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

14.2.8 HAL_DMA_Start_IT

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

14.2.9 HAL_DMA_Abort

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

Notes

- 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.

14.2.10 HAL_DMA_PollForTransfer

Function Name	HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, uint32_t CompleteLevel, uint32_t Timeout)
Function Description	Polling for transfer complete.
Parameters	<ul style="list-style-type: none"> hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

- **CompleteLevel:** Specifies the DMA level complete.
 - **Timeout:** Timeout duration.
- Return values
- HAL status

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

14.2.12 HAL_DMA_GetState

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

14.2.13 HAL_DMA_GetError

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

14.3 DMA Firmware driver defines

14.3.1 DMA

DMA Data transfer direction

<code>DMA_PERIPH_TO_MEMORY</code>	Peripheral to memory direction
<code>DMA_MEMORY_TO_PERIPH</code>	Memory to peripheral direction
<code>DMA_MEMORY_TO_MEMORY</code>	Memory to memory direction

DMA Error Code

<code>HAL_DMA_ERROR_NONE</code>	No error
<code>HAL_DMA_ERROR_TE</code>	Transfer error
<code>HAL_DMA_ERROR_TIMEOUT</code>	Timeout error

DMA Exported Macros



<code>__HAL_DMARESET_HANDLESTATE</code>	Description: <ul style="list-style-type: none">• Reset DMA handle state. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: DMA handle. Return value: <ul style="list-style-type: none">• None
<code>__HAL_DMA_ENABLE</code>	Description: <ul style="list-style-type: none">• Enable the specified DMA Channel. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: DMA handle Return value: <ul style="list-style-type: none">• None
<code>__HAL_DMA_DISABLE</code>	Description: <ul style="list-style-type: none">• Disable the specified DMA Channel. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: DMA handle Return value: <ul style="list-style-type: none">• None
<code>__HAL_DMA_ENABLE_IT</code>	Description: <ul style="list-style-type: none">• Enables the specified DMA Channel interrupts. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: DMA handle• <code>__INTERRUPT__</code>: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:<ul style="list-style-type: none">– <code>DMA_IT_TC</code>: Transfer complete interrupt mask– <code>DMA_IT_HT</code>: Half transfer complete interrupt mask– <code>DMA_IT_TE</code>: Transfer error interrupt mask Return value: <ul style="list-style-type: none">• None
<code>__HAL_DMA_DISABLE_IT</code>	Description: <ul style="list-style-type: none">• Disables the specified DMA Channel interrupts. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: 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 is enabled or disabled.

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

<code>DMA_FLAG_GL1</code>	Channel 1 global interrupt flag
<code>DMA_FLAG_TC1</code>	Channel 1 transfer complete flag
<code>DMA_FLAG_HT1</code>	Channel 1 half transfer flag
<code>DMA_FLAG_TE1</code>	Channel 1 transfer error flag
<code>DMA_FLAG_GL2</code>	Channel 2 global interrupt flag
<code>DMA_FLAG_TC2</code>	Channel 2 transfer complete flag
<code>DMA_FLAG_HT2</code>	Channel 2 half transfer flag
<code>DMA_FLAG_TE2</code>	Channel 2 transfer error flag
<code>DMA_FLAG_GL3</code>	Channel 3 global interrupt flag
<code>DMA_FLAG_TC3</code>	Channel 3 transfer complete flag
<code>DMA_FLAG_HT3</code>	Channel 3 half transfer flag
<code>DMA_FLAG_TE3</code>	Channel 3 transfer error flag
<code>DMA_FLAG_GL4</code>	Channel 4 global interrupt flag

DMA_FLAG_TC4	Channel 4 transfer complete flag
DMA_FLAG_HT4	Channel 4 half transfer flag
DMA_FLAG_TE4	Channel 4 transfer error flag
DMA_FLAG_GL5	Channel 5 global interrupt flag
DMA_FLAG_TC5	Channel 5 transfer complete flag
DMA_FLAG_HT5	Channel 5 half transfer flag
DMA_FLAG_TE5	Channel 5 transfer error flag
DMA_FLAG_GL6	Channel 6 global interrupt flag
DMA_FLAG_TC6	Channel 6 transfer complete flag
DMA_FLAG_HT6	Channel 6 half transfer flag
DMA_FLAG_TE6	Channel 6 transfer error flag
DMA_FLAG_GL7	Channel 7 global interrupt flag
DMA_FLAG_TC7	Channel 7 transfer complete flag
DMA_FLAG_HT7	Channel 7 half transfer flag
DMA_FLAG_TE7	Channel 7 transfer error flag

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

DMA Memory incremented mode

DMA_MINC_ENABLE Memory increment mode Enable

DMA_MINC_DISABLE Memory increment mode Disable

DMA mode

DMA_NORMAL Normal Mode

DMA_CIRCULAR Circular Mode

DMA Peripheral data size

DMA_PDATAALIGN_BYTE Peripheral data alignment : Byte

DMA_PDATAALIGN_HALFWORD Peripheral data alignment : HalfWord

DMA_PDATAALIGN_WORD Peripheral data alignment : Word

DMA Peripheral incremented mode

DMA_PINC_ENABLE Peripheral increment mode Enable

DMA_PINC_DISABLE Peripheral increment mode Disable

DMA Priority level

DMA_PRIORITY_LOW	Priority level : Low
DMA_PRIORITY_MEDIUM	Priority level : Medium
DMA_PRIORITY_HIGH	Priority level : High
DMA_PRIORITY VERY HIGH	Priority level : Very_High

15 HAL DMA Extension Driver

15.1 DMAEx Firmware driver defines

15.1.1 DMAEx

DMAEx Exported Constants

DMA1_CHANNEL1_RMP	Internal define for remapping on STM32F09x/30xC
DMA1_CHANNEL2_RMP	Internal define for remapping on STM32F09x/30xC
DMA1_CHANNEL3_RMP	Internal define for remapping on STM32F09x/30xC
DMA1_CHANNEL4_RMP	Internal define for remapping on STM32F09x/30xC
DMA1_CHANNEL5_RMP	Internal define for remapping on STM32F09x/30xC
DMA1_CHANNEL6_RMP	Internal define for remapping on STM32F09x/30xC
DMA1_CHANNEL7_RMP	Internal define for remapping on STM32F09x/30xC
DMA2_CHANNEL1_RMP	Internal define for remapping on STM32F09x/30xC
DMA2_CHANNEL2_RMP	Internal define for remapping on STM32F09x/30xC
DMA2_CHANNEL3_RMP	Internal define for remapping on STM32F09x/30xC
DMA2_CHANNEL4_RMP	Internal define for remapping on STM32F09x/30xC
DMA2_CHANNEL5_RMP	Internal define for remapping on STM32F09x/30xC
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_TCx`: Transfer complete flag
 - `DMA_FLAG_HTx`: Half transfer complete flag
 - `DMA_FLAG_TEx`: 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_TCx: Transfer complete flag
 - DMA_FLAG_HTx: Half transfer complete flag
 - DMA_FLAG_TEx: 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_ProcessTypeDef

Data Fields

- `__IO FLASH_ProcedureTypeDef ProcedureOnGoing`
- `__IO uint32_t DataRemaining`
- `__IO uint32_t Address`
- `__IO uint64_t Data`
- `HAL_LockTypeDef Lock`
- `__IO uint32_t ErrorCode`

Field Documentation

- `__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing`
Internal variable to indicate which procedure is ongoing or not in IT context
- `__IO uint32_t FLASH_ProcessTypeDef::DataRemaining`
Internal variable to save the remaining pages to erase or half-word to program in IT context
- `__IO uint32_t FLASH_ProcessTypeDef::Address`
Internal variable to save address selected for program or erase
- `__IO uint64_t FLASH_ProcessTypeDef::Data`
Internal variable to save data to be programmed
- `HAL_LockTypeDef FLASH_ProcessTypeDef::Lock`
FLASH locking object
- `__IO uint32_t FLASH_ProcessTypeDef::ErrorCode`
FLASH error code This parameter can be a value of [FLASH_Error_Codes](#)

16.2 FLASH Firmware driver API description

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.

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. FLASH Option Bytes Programming functions: this group includes all needed functions to manage the Option Bytes:
 - Lock and Unlock the Option Bytes
 - Set/Reset the write protection
 - Set the Read protection Level
 - Program the user Option Bytes
 - Launch the Option Bytes loader
 - Erase Option Bytes
 - Program the data Option Bytes
 - Get the Write protection.
 - Get the user option bytes.
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 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

This section contains the following APIs:

- [`HAL_FLASH_Unlock\(\)`](#)
- [`HAL_FLASH_Lock\(\)`](#)
- [`HAL_FLASH_OB_Unlock\(\)`](#)
- [`HAL_FLASH_OB_Lock\(\)`](#)
- [`HAL_FLASH_OB_Launch\(\)`](#)

16.2.4 Peripheral State functions

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

This section contains the following APIs:

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

16.2.5 HAL_FLASH_Program

Function Name	HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint64_t Data)
Function Description	Program halfword, word or double word at a specified address.
Parameters	<ul style="list-style-type: none"> TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program Address: Specifies the address to be programmed. Data: Specifies the data to be programmed
Return values	<ul style="list-style-type: none"> HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> 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 If an erase and a program operations are requested simultaneously, the erase operation is performed before the program one. FLASH should be previously erased before new programmation (only exception to this is when 0x0000 is programmed)

16.2.6 HAL_FLASH_Program_IT

Function Name	HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t Data)
Function Description	Program halfword, word or double word at a specified address with interrupt enabled.
Parameters	<ul style="list-style-type: none"> TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program Address: Specifies the address to be programmed. Data: Specifies the data to be programmed
Return values	<ul style="list-style-type: none"> HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> 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 If an erase and a program operations are requested simultaneously, the erase operation is performed before the program one.

16.2.7 HAL_FLASH_IRQHandler

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

16.2.8 HAL_FLASH_EndOfOperationCallback

Function Name	void HAL_FLASH_EndOfOperationCallback (uint32_t
---------------	--

ReturnValue

Function Description	FLASH end of operation interrupt callback.
Parameters	<ul style="list-style-type: none"> • ReturnValue: The value saved in this parameter depends on the ongoing procedureMass Erase: No return value expectedPages Erase: Address of the page which has been erasedProgram: Address which was selected for data program
Return values	<ul style="list-style-type: none"> • none

16.2.9 HAL_FLASH_OperationErrorCallback

Function Name	void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)
Function Description	FLASH operation error interrupt callback.
Parameters	<ul style="list-style-type: none"> • ReturnValue: The value saved in this parameter depends on the ongoing procedureMass Erase: No return value expectedPages Erase: Address of the page which returned an errorProgram: Address which was selected for data program
Return values	<ul style="list-style-type: none"> • none

16.2.10 HAL_FLASH_Unlock

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

16.2.11 HAL_FLASH_Lock

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

16.2.12 HAL_FLASH_OB_Unlock

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

16.2.13 HAL_FLASH_OB_Lock

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

16.2.14 HAL_FLASH_OB_Launch

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)
Function Description	Launch the option byte loading.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> • This function will reset automatically the MCU.

16.2.15 HAL_FLASH_GetError

Function Name	uint32_t HAL_FLASH_GetError (void)
Function Description	Get the specific FLASH error flag.
Return values	<ul style="list-style-type: none"> • FLASH_ErrorCode The returned value can be: FLASH Error Codes

16.3 FLASH Firmware driver defines

16.3.1 FLASH

FLASH Error Codes

HAL_FLASH_ERROR_NONE
 HAL_FLASH_ERROR_PROG
 HAL_FLASH_ERROR_WRP

FLASH Flag definition

FLASH_FLAG_BSY FLASH Busy flag
 FLASH_FLAG_PGERR FLASH Programming error flag
 FLASH_FLAG_WRPERR FLASH Write protected error flag
 FLASH_FLAG_EOP FLASH End of Operation flag

FLASH Interrupts

<u>__HAL_FLASH_ENABLE_IT</u>	Description:
	<ul style="list-style-type: none"> • Enable the specified FLASH interrupt.
	Parameters:
	<ul style="list-style-type: none"> • __INTERRUPT__: : FLASH interrupt This parameter can be any combination of the following values: <ul style="list-style-type: none"> - FLASH_IT_EOP: End of FLASH Operation Interrupt - FLASH_IT_ERR: Error Interrupt
	Return value:
	<ul style="list-style-type: none"> • none
<u>__HAL_FLASH_DISABLE_IT</u>	Description:
	<ul style="list-style-type: none"> • Disable the specified FLASH interrupt.
	Parameters:
	<ul style="list-style-type: none"> • __INTERRUPT__: : FLASH interrupt This parameter can be any combination of the following values: <ul style="list-style-type: none"> - FLASH_IT_EOP: End of FLASH Operation

- Interrupt
 - FLASH_IT_ERR: Error Interrupt

Return value:

- none

_HAL_FLASH_GET_FLAG

- Get the specified FLASH flag status.

Parameters:

- __FLAG__: specifies the FLASH flag to check. This parameter can be one of the following values:
 - FLASH_FLAG_BSY : FLASH Busy flag
 - 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:

- 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_BSY : FLASH Busy flag
 - 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

_HAL_FLASH_SET_LATENCY**Description:**

- Set the FLASH Latency.

Parameters:

- `__LATENCY__`: FLASH Latency The value of this parameter depend on device used within the same series

Return value:

- None

`__HAL_FLASH_GET_LATENCY`**Description:**

- Get the FLASH Latency.

Return value:

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

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 Type Program`FLASH_TYPEPROGRAM_HALFWORD`

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

`FLASH_TYPEPROGRAM_WORD`

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

`FLASH_TYPEPROGRAM_DOUBLEWORD`

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

17 HAL FLASH Extension Driver

17.1 FLASHEx Firmware driver registers structures

17.1.1 FLASH_EraseInitTypeDef

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 [**FLASHEx_Type_Erase**](#)
- *uint32_t FLASH_EraseInitTypeDef::PageAddress*
PageAddress: Initial FLASH page address to erase when mass erase is disabled. This parameter must be a number between Min_Data = FLASH_BASE and Max_Data = FLASH_BANK1_END
- *uint32_t FLASH_EraseInitTypeDef::NbPages*
NbPages: Number of pages to be erased. This parameter must be a value between Min_Data = 1 and Max_Data = (max number of pages - value of initial page)

17.1.2 FLASH_OBProgramInitTypeDef

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 [**FLASHEx_OB_Type**](#)
- *uint32_t FLASH_OBProgramInitTypeDef::WRPState*
WRPState: Write protection activation or deactivation. This parameter can be a value of [**FLASHEx_OB_WRP_State**](#)
- *uint32_t FLASH_OBProgramInitTypeDef::WRPPage*
WRPPage: specifies the page(s) to be write protected. This parameter can be a value of [**FLASHEx_OB_Write_Protection**](#)

- ***uint8_t FLASH_OBProgramInitTypeDef::RDPLevel***
RDPLevel: Set the read protection level.. This parameter can be a value of ***FLASHEx_OB_Read_Protection***
- ***uint8_t FLASH_OBProgramInitTypeDef::USERConfig***
USERConfig: Program the FLASH User Option Byte: IWDG / STOP / STDBY / BOOT1 / VDDA_ANALOG / SRAM_PARITY This parameter can be a combination of ***FLASHEx_OB_Watchdog***, ***FLASHEx_OB_nRST_STOP***, ***FLASHEx_OB_nRST_STDBY***, ***FLASHEx_OB_BOOT1***, ***FLASHEx_OB_VDDA_Analog_Monitoring*** and ***FLASHEx_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 ***FLASHEx_OB_Data_Address***
- ***uint8_t FLASH_OBProgramInitTypeDef::DATAData***
DATAData: Data to be stored in the option byte DATA This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF

17.2 FLASHEx Firmware driver API description

17.2.1 IO operation functions

This section contains the following APIs:

- ***HAL_FLASHEx_Erase()***
- ***HAL_FLASHEx_Erase_IT()***

17.2.2 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

This section contains the following APIs:

- ***HAL_FLASHEx_OBErase()***
- ***HAL_FLASHEx_OBProgram()***
- ***HAL_FLASHEx_OBGetConfig()***

17.2.3 HAL_FLASHEx_Erase

Function Name	HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraselInitTypeDef * pEraselInit, uint32_t * PageError)
Function Description	Perform a mass erase or erase the specified FLASH memory pages.
Parameters	<ul style="list-style-type: none"> • pEraselInit: pointer to an FLASH_EraselInitTypeDef structure that contains the configuration information for the erasing. • PageError: 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)
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> • 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

17.2.4 HAL_FLASHEx_Erase_IT

Function Name	HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)
Function Description	Perform a mass erase or erase the specified FLASH memory sectors with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • pEraseInit: pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> • 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

17.2.5 HAL_FLASHEx_OBErase

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

17.2.6 HAL_FLASHEx_OBProgram

Function Name	HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)
Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none"> • pOBInit: pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> • 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)

17.2.7 HAL_FLASHEx_OBGetConfig

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

contains the configuration information for the programming.

Return values • None

17.3 FLASHEx Firmware driver defines

17.3.1 FLASHEx

FLASHEx OB BOOT0

OB_BOOT0_RESET BOOT0 Reset
OB_BOOT0_SET BOOT0 Set

FLASH OB BOOT1

OB_BOOT1_RESET BOOT1 Reset
OB_BOOT1_SET BOOT1 Set

FLASHEx OB BOOT SEL

OB_BOOT_SEL_RESET BOOT_SEL Reset
OB_BOOT_SEL_SET BOOT_SEL Set

Option Byte Data Address

OB_DATA_ADDRESS_DATA0
OB_DATA_ADDRESS_DATA1

FLASH OB nRST STDBY

OB_STDBY_NO_RST No reset generated when entering in STANDBY
OB_STDBY_RST Reset generated when entering in STANDBY

FLASH OB nRST STOP

OB_STOP_NO_RST No reset generated when entering in STOP
OB_STOP_RST Reset generated when entering in STOP

FLASH OB RAM Parity Check Enable

OB_RAM_PARITY_CHECK_SET RAM parity check enable set
OB_RAM_PARITY_CHECK_RESET RAM parity check enable reset

FLASH OB Read Protection

OB_RDP_LEVEL_0
OB_RDP_LEVEL_1
OB_RDP_LEVEL_2 Warning: When enabling read protection level 2 it's no more possible to go back to level 1 or 0

FLASH Option Bytes Type

OPTIONBYTE_WRP WRP option byte configuration
OPTIONBYTE_RDP RDP option byte configuration
OPTIONBYTE_USER USER option byte configuration
OPTIONBYTE_DATA DATA option byte configuration

FLASH OB VDDA Analog Monitoring

OB_VDDA_ANALOG_ON Analog monitoring on VDDA Power source ON
OB_VDDA_ANALOG_OFF Analog monitoring on VDDA Power source OFF

FLASH OB Watchdog

OB_IWDG_SW Software WDG selected
OB_IWDG_HW Hardware WDG selected

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

FLASH WRP State

OB_WRPSTATE_DISABLE Disable the write protection of the desired pages

OB_WRPSTATE_ENABLE Enable the write protection of the desired pages

FLASHEx Page Size

FLASH_PAGE_SIZE

FLASH Type Erase

FLASH_TYPEERASE_PAGES Pages erase only

FLASH_TYPEERASE_MASSERASE Flash mass erase activation

18 HAL GPIO Generic Driver

18.1 GPIO Firmware driver registers structures

18.1.1 GPIO_InitTypeDef

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

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 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 (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 :
`_HAL_RCC_GPIOx_CLK_ENABLE()`.
2. Configure the GPIO pin(s) using `HAL_GPIO_Init()`.
 - Configure the IO mode using "Mode" member from `GPIO_InitTypeDef` structure
 - Activate Pull-up, Pull-down resistor using "Pull" member from `GPIO_InitTypeDef` structure.
 - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from `GPIO_InitTypeDef` structure.
 - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from `GPIO_InitTypeDef` structure.
 - Analog mode is required when a pin is to be used as ADC channel or DAC output.
 - In case of external interrupt/event selection the "Mode" member from `GPIO_InitTypeDef` structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using `HAL_NVIC_SetPriority()` and enable it using `HAL_NVIC_EnableIRQ()`.
4. `HAL_GPIO_DeInit` allows to set register values to their reset value. It's also recommended 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 PF0 and PF1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

18.2.3 Initialization and de-initialization functions

This section contains the following APIs:

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

18.2.4 IO operation functions

This section contains the following APIs:

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

18.2.5 HAL_GPIO_Init

Function Name	<code>void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_InitStruct)</code>
Function Description	Initialize the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..F) to select the GPIO peripheral for STM32F0 family • GPIO_InitStruct: pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	<ul style="list-style-type: none"> • None

18.2.6 HAL_GPIO_DeInit

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

18.2.7 HAL_GPIO_ReadPin

Function Name	<code>GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</code>
Function Description	Read the specified input port pin.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..F) to select the GPIO peripheral for STM32F0 family • GPIO_Pin: specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).
Return values	<ul style="list-style-type: none"> • The input port pin value.

18.2.8 HAL_GPIO_WritePin

Function Name	<code>void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)</code>
Function Description	Set or clear the selected data port bit.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..H) to select the GPIO peripheral for STM32F0 family • GPIO_Pin: specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). • PinState: specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: GPIO_PIN_RESET: to clear the port pinGPIO_PIN_SET: to set the port pin
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This function uses GPIOx_BSRR and GPIOx_BRR registers to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

18.2.9 HAL_GPIO_TogglePin

Function Name	<code>void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</code>
Function Description	Toggle the specified GPIO pin.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..F) to select the GPIO peripheral for STM32F0 family • GPIO_Pin: specifies the pin to be toggled.
Return values	<ul style="list-style-type: none"> • None

18.2.10 HAL_GPIO_LockPin

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

18.2.11 HAL_GPIO_EXTI_IRQHandler

Function Name	void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)
Function Description	Handle EXTI interrupt request.
Parameters	<ul style="list-style-type: none"> • GPIO_Pin: Specifies the port pin connected to corresponding EXTI line.
Return values	<ul style="list-style-type: none"> • None

18.2.12 HAL_GPIO_EXTI_Callback

Function Name	void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)
Function Description	EXTI line detection callback.
Parameters	<ul style="list-style-type: none"> • GPIO_Pin: Specifies the port pin connected to corresponding EXTI line.
Return values	<ul style="list-style-type: none"> • None

18.3 GPIO Firmware driver defines

18.3.1 GPIO

GPIO Exported Macros

`_HAL_GPIO_EXTI_GET_FLAG`

Description:

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

Parameters:

- `_EXTI_LINE_`: specifies the EXTI line flag to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

Return value:

- The new state of `_EXTI_LINE_` (SET or RESET).

`_HAL_GPIO_EXTI_CLEAR_FLAG`

Description:

- Clear the EXTI's line pending flags.

Parameters:

- `_EXTI_LINE_`: specifies the EXTI lines flags to clear. This parameter can be any combination of `GPIO_PIN_x` where x can be (0..15)

Return value:

- None

`_HAL_GPIO_EXTI_GET_IT`

Description:

- Check whether the specified EXTI line is asserted or not.

Parameters:

- `_EXTI_LINE_`: specifies the EXTI line to

check. This parameter can be
GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

`__HAL_GPIO_EXTI_CLEAR_IT`

Description:

- Clear the EXTI's line pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

`__HAL_GPIO_EXTI_GENERATE_SWIT`

Description:

- Generate 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

GPIO pins

GPIO_PIN_0
GPIO_PIN_1
GPIO_PIN_2
GPIO_PIN_3
GPIO_PIN_4
GPIO_PIN_5
GPIO_PIN_6
GPIO_PIN_7
GPIO_PIN_8
GPIO_PIN_9
GPIO_PIN_10
GPIO_PIN_11
GPIO_PIN_12
GPIO_PIN_13
GPIO_PIN_14
GPIO_PIN_15
GPIO_PIN_All
GPIO_PIN_MASK

GPIO pull

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

GPIO speed

GPIO_SPEED_LOW Low speed
GPIO_SPEED_MEDIUM Medium speed
GPIO_SPEED_HIGH High speed

19 HAL GPIO Extension Driver

19.1 GPIOEx Firmware driver defines

19.1.1 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	
GPIOEx_Get Port Index	
GPIO_GET_INDEX	

20 HAL I2C Generic Driver

20.1 I2C Firmware driver registers structures

20.1.1 I2C_InitTypeDef

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

Data Fields

- *I2C_TypeDef * Instance*
- *I2C_InitTypeDef Init*
- *uint8_t * pBuffPtr*
- *uint16_t XferSize*
- *_IO uint16_t XferCount*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *_IO HAL_I2C_StateTypeDef State*
- *_IO uint32_t ErrorCode*

Field Documentation

- *I2C_TypeDef* I2C_HandleTypeDef::Instance*
I2C registers base address
- *I2C_InitTypeDef I2C_HandleTypeDef::Init*
I2C communication parameters
- *uint8_t* I2C_HandleTypeDef::pBuffPtr*
Pointer to I2C transfer buffer
- *uint16_t I2C_HandleTypeDef::XferSize*
I2C transfer size
- *_IO uint16_t I2C_HandleTypeDef::XferCount*
I2C transfer counter
- *DMA_HandleTypeDef* I2C_HandleTypeDef::hdmatx*
I2C Tx DMA handle parameters
- *DMA_HandleTypeDef* I2C_HandleTypeDef::hdmarx*
I2C Rx DMA handle parameters
- *HAL_LockTypeDef I2C_HandleTypeDef::Lock*
I2C locking object
- *_IO HAL_I2C_StateTypeDef I2C_HandleTypeDef::State*
I2C communication state
- *_IO uint32_t I2C_HandleTypeDef::ErrorCode*
I2C Error code

20.2 I2C Firmware driver API description

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 implementing the HAL_I2C_MspInit() API:
 - a. Enable the I2Cx interface clock
 - b. I2C pins configuration
 - Enable the clock for the I2C GPIOs
 - Configure I2C pins as alternate function open-drain
 - c. NVIC configuration if you need to use interrupt process
 - Configure the I2Cx interrupt priority
 - Enable the NVIC I2C IRQ Channel
 - d. DMA Configuration if you need to use DMA process

- Declare a DMA_HandleTypeDef handle structure for the transmit or receive channel
 - Enable the DMAx interface clock using
 - Configure the DMA handle parameters
 - Configure the DMA Tx or Rx channel
 - Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel
3. Configure the Communication 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(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I2C_MsplInit(&hi2c) API.
 5. To check if target device is ready for communication, use the function HAL_I2C_IsDeviceReady()
 6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Receive_IT()
- At reception end of transfer HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Transmit_IT()

- At transmission end of transfer HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()

Interrupt mode IO MEM operation

- Write an amount of data in non-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 non-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 non-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 non-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At MEM end of read transfer HAL_I2C_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()

I2C HAL driver macros list

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

- __HAL_I2C_ENABLE: Enable the I2C peripheral
- __HAL_I2C_DISABLE: Disable the I2C peripheral
- __HAL_I2C_GET_FLAG: Checks whether the specified I2C flag is set or not
- __HAL_I2C_CLEAR_FLAG: Clear the specified I2C pending flag
- __HAL_I2C_ENABLE_IT: Enable the specified I2C interrupt
- __HAL_I2C_DISABLE_IT: Disable the specified I2C interrupt



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

20.2.2 Initialization and de-initialization functions

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

- User must Implement HAL_I2C_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2C_Init() to configure the selected device with the selected configuration:
 - 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.

This section contains the following APIs:

- [**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.
 - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
 - [*HAL_I2C_Master_Transmit\(\)*](#)
 - [*HAL_I2C_Master_Receive\(\)*](#)
 - [*HAL_I2C_Slave_Transmit\(\)*](#)
 - [*HAL_I2C_Slave_Receive\(\)*](#)
 - [*HAL_I2C_Mem_Write\(\)*](#)
 - [*HAL_I2C_Mem_Read\(\)*](#)
 - [*HAL_I2C_IsDeviceReady\(\)*](#)
3. No-Blocking mode functions with Interrupt are :
 - [*HAL_I2C_Master_Transmit_IT\(\)*](#)
 - [*HAL_I2C_Master_Receive_IT\(\)*](#)
 - [*HAL_I2C_Slave_Transmit_IT\(\)*](#)
 - [*HAL_I2C_Slave_Receive_IT\(\)*](#)
 - [*HAL_I2C_Mem_Write_IT\(\)*](#)
 - [*HAL_I2C_Mem_Read_IT\(\)*](#)
4. No-Blocking mode functions with DMA are :
 - [*HAL_I2C_Master_Transmit_DMA\(\)*](#)
 - [*HAL_I2C_Master_Receive_DMA\(\)*](#)
 - [*HAL_I2C_Slave_Transmit_DMA\(\)*](#)
 - [*HAL_I2C_Slave_Receive_DMA\(\)*](#)
 - [*HAL_I2C_Mem_Write_DMA\(\)*](#)
 - [*HAL_I2C_Mem_Read_DMA\(\)*](#)
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - [*HAL_I2C_MemTxCpltCallback\(\)*](#)
 - [*HAL_I2C_MemRxCpltCallback\(\)*](#)
 - [*HAL_I2C_MasterTxCpltCallback\(\)*](#)
 - [*HAL_I2C_MasterRxCpltCallback\(\)*](#)
 - [*HAL_I2C_SlaveTxCpltCallback\(\)*](#)
 - [*HAL_I2C_SlaveRxCpltCallback\(\)*](#)
 - [*HAL_I2C_ErrorCallback\(\)*](#)

This section contains the following APIs:

- [*HAL_I2C_Master_Transmit\(\)*](#)
- [*HAL_I2C_Master_Receive\(\)*](#)
- [*HAL_I2C_Slave_Transmit\(\)*](#)
- [*HAL_I2C_Slave_Receive\(\)*](#)
- [*HAL_I2C_Master_Transmit_IT\(\)*](#)
- [*HAL_I2C_Master_Receive_IT\(\)*](#)
- [*HAL_I2C_Slave_Transmit_IT\(\)*](#)

- [*HAL_I2C_Slave_Receive_IT\(\)*](#)
- [*HAL_I2C_Master_Transmit_DMA\(\)*](#)
- [*HAL_I2C_Master_Receive_DMA\(\)*](#)
- [*HAL_I2C_Slave_Transmit_DMA\(\)*](#)
- [*HAL_I2C_Slave_Receive_DMA\(\)*](#)
- [*HAL_I2C_Mem_Write\(\)*](#)
- [*HAL_I2C_Mem_Read\(\)*](#)
- [*HAL_I2C_Mem_Write_IT\(\)*](#)
- [*HAL_I2C_Mem_Read_IT\(\)*](#)
- [*HAL_I2C_Mem_Write_DMA\(\)*](#)
- [*HAL_I2C_Mem_Read_DMA\(\)*](#)
- [*HAL_I2C_IsDeviceReady\(\)*](#)

20.2.4 Peripheral State and Errors functions

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

This section contains the following APIs:

- [*HAL_I2C_GetState\(\)*](#)
- [*HAL_I2C_GetError\(\)*](#)

20.2.5 HAL_I2C_Init

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

20.2.6 HAL_I2C_DeInit

Function Name	HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)
Function Description	Deinitialize the I2C peripheral.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • HAL status

20.2.7 HAL_I2C_MspInit

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

20.2.8 HAL_I2C_MspDeInit

Function Name	void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)
---------------	--

Function Description	Deinitialize the I2C MSP.
Parameters	<ul style="list-style-type: none"> hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> None

20.2.9 HAL_I2C_Master_Transmit

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits in master mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. DevAddress: Target device address pData: Pointer to data buffer Size: Amount of data to be sent Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

20.2.10 HAL_I2C_Master_Receive

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Receive(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receives in master mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. DevAddress: Target device address pData: Pointer to data buffer Size: Amount of data to be sent Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

20.2.11 HAL_I2C_Slave_Transmit

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Transmit(I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits in slave mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. pData: Pointer to data buffer Size: Amount of data to be sent Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

20.2.12 HAL_I2C_Slave_Receive

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive in slave mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. pData: Pointer to data buffer Size: Amount of data to be sent Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

20.2.13 HAL_I2C_Master_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)
Function Description	Transmit in master mode an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. DevAddress: Target device address pData: Pointer to data buffer Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status

20.2.14 HAL_I2C_Master_Receive_IT

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)
Function Description	Receive in master mode an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. DevAddress: Target device address pData: Pointer to data buffer Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status

20.2.15 HAL_I2C_Slave_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)
Function Description	Transmit in slave mode an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. pData: Pointer to data buffer

- **Size:** Amount of data to be sent
- HAL status

20.2.16 HAL_I2C_Slave_Receive_IT

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)
Function Description	Receive in slave mode an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

20.2.17 HAL_I2C_Master_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)
Function Description	Transmit in master mode an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

20.2.18 HAL_I2C_Master_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)
Function Description	Receive in master mode an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

20.2.19 HAL_I2C_Slave_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)
Function Description	Transmit in slave mode an amount of data in non-blocking mode

with DMA.

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

20.2.20 HAL_I2C_Slave_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)
Function Description	Receive in slave mode an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. pData: Pointer to data buffer Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status

20.2.21 HAL_I2C_Mem_Write

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

20.2.22 HAL_I2C_Mem_Read

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

- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- HAL status

20.2.23 HAL_I2C_Mem_Write_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)</code>
Function Description	Write an amount of data in non-blocking mode with Interrupt to a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • MemAddress: Internal memory address • MemAddSize: Size of internal memory address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

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 non-blocking mode with Interrupt from a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • MemAddress: Internal memory address • MemAddSize: Size of internal memory address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

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 non-blocking mode with DMA to a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that

- contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
 - **MemAddress:** Internal memory address
 - **MemAddSize:** Size of internal memory address
 - **pData:** Pointer to data buffer
 - **Size:** Amount of data to be sent
- Return values**
- HAL status

20.2.26 HAL_I2C_Mem_Read_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)</code>
Function Description	Reads an amount of data in non-blocking mode with DMA from a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • MemAddress: Internal memory address • MemAddSize: Size of internal memory address • pData: Pointer to data buffer • Size: Amount of data to be read
Return values	<ul style="list-style-type: none"> • HAL status

20.2.27 HAL_I2C_IsDeviceReady

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

Notes

- This function is used with Memory devices

20.2.28 HAL_I2C_EV_IRQHandler

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

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

20.2.29 HAL_I2C_ER_IRQHandler

Function Name	void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)
Function Description	This function handles I2C error interrupt request.
Parameters	<ul style="list-style-type: none">hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none">None

20.2.30 HAL_I2C_MasterTxCpltCallback

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

20.2.31 HAL_I2C_MasterRxCpltCallback

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

20.2.32 HAL_I2C_SlaveTxCpltCallback

Function Name	void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Slave Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none">None

20.2.33 HAL_I2C_SlaveRxCpltCallback

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

20.2.34 HAL_I2C_MemTxCpltCallback

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

20.2.35 HAL_I2C_MemRxCpltCallback

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

20.2.36 HAL_I2C_ErrorCallback

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

20.2.37 HAL_I2C_GetState

Function Name	HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)
Function Description	Return the I2C handle state.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • HAL state

20.2.38 HAL_I2C_GetError

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

20.3 I2C Firmware driver defines

20.3.1 I2C

I2C addressing mode

`I2C_ADDRESSINGMODE_7BIT`
`I2C_ADDRESSINGMODE_10BIT`

I2C dual addressing mode

`I2C_DUALADDRESS_DISABLE`
`I2C_DUALADDRESS_ENABLE`

I2C Error Code definition

<code>HAL_I2C_ERROR_NONE</code>	No error
<code>HAL_I2C_ERROR_BERR</code>	BERR error
<code>HAL_I2C_ERROR_ARLO</code>	ARLO error
<code>HAL_I2C_ERROR_AF</code>	ACKF error
<code>HAL_I2C_ERROR_OVR</code>	OVR error
<code>HAL_I2C_ERROR_DMA</code>	DMA transfer error
<code>HAL_I2C_ERROR_TIMEOUT</code>	Timeout error
<code>HAL_I2C_ERROR_SIZE</code>	Size Management error

I2C Exported Macros

<code>_HAL_I2C_RESET_HANDLE_STATE</code>	Description:
	<ul style="list-style-type: none"> • Reset I2C handle state.
	Parameters:
	<ul style="list-style-type: none"> • <code>_HANDLE_</code>: specifies the I2C Handle.
	Return value:
	<ul style="list-style-type: none"> • None
<code>_HAL_I2C_ENABLE_IT</code>	Description:
	<ul style="list-style-type: none"> • Enable the specified I2C interrupt.
	Parameters:
	<ul style="list-style-type: none"> • <code>_HANDLE_</code>: specifies the I2C Handle. • <code>_INTERRUPT_</code>: specifies the interrupt source to enable. 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:

- None

__HAL_I2C_DISABLE_IT

- Disable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. 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:

- None

__HAL_I2C_GET_IT_SOURCE

- Check whether the specified I2C interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the I2C interrupt source to check. This parameter can be one of the following values:
 - I2C_IT_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

- 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 __INTERRUPT__ (TRUE or FALSE).

I2C_FLAG_MASK**Description:**

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

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - I2C_FLAG_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_GET_FLAG**__HAL_I2C_CLEAR_FLAG****Description:**

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

Parameters:

- `__HANDLE__`: specifies the I2C Handle.
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
 - `I2C_FLAG_ADDR`: Address matched (slave mode)
 - `I2C_FLAG_AF`: Acknowledge failure received 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`

Description:

- Enable the specified I2C peripheral.

Parameters:

- `__HANDLE__`: specifies the I2C Handle.

Return value:

- None

`__HAL_I2C_DISABLE`

Description:

- Disable the specified I2C peripheral.

Parameters:

- `__HANDLE__`: specifies the I2C Handle.

Return value:

- None

I2C Flag definition

`I2C_FLAG_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_DISABLE
I2C_GENERALCALL_ENABLE

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

I2C nostretch mode

I2C_NOSTRETCH_DISABLE
I2C_NOSTRETCH_ENABLE

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

I2C ReloadEndMode definition

I2C_RELOAD_MODE

I2C_AUTOEND_MODE

I2C_SOFTEND_MODE

I2C StartStopMode definition

I2C_NO_STARTSTOP

I2C_GENERATE_STOP

I2C_GENERATE_START_READ

I2C_GENERATE_START_WRITE

21 HAL I2C Extension Driver

21.1 I2CEEx Firmware driver API description

21.1.1 I2C peripheral Extended 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_ConfigAnalogFilter()`
2. Configure I2C Digital noise filter using the function `HAL_I2CEEx_ConfigDigitalFilter()`
3. Configure the enable or disable of I2C Wake Up Mode using the functions :
 - `HAL_I2CEEx_EnableWakeUp()`
 - `HAL_I2CEEx_DisableWakeUp()`
4. Configure the enable or disable of fast mode plus driving capability using the functions :
 - `HAL_I2CEEx_EnableFastModePlus()`
 - `HAL_I2CEEx_DisableFastModePlus()`

21.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:

- [`HAL_I2CEEx_ConfigAnalogFilter\(\)`](#)
- [`HAL_I2CEEx_ConfigDigitalFilter\(\)`](#)
- [`HAL_I2CEEx_EnableWakeUp\(\)`](#)
- [`HAL_I2CEEx_DisableWakeUp\(\)`](#)
- [`HAL_I2CEEx_EnableFastModePlus\(\)`](#)
- [`HAL_I2CEEx_DisableFastModePlus\(\)`](#)

21.1.4 `HAL_I2CEEx_ConfigAnalogFilter`

Function Name `HAL_StatusTypeDef HAL_I2CEEx_ConfigAnalogFilter(I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)`

Function Description Configures I2C Analog noise filter.

Parameters

- **hi2c:** : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **AnalogFilter:** : new state of the Analog filter.

Return values

- HAL status

21.1.5 HAL_I2CEEx_ConfigDigitalFilter

Function Name	<code>HAL_StatusTypeDef HAL_I2CEEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)</code>
Function Description	Configures I2C Digital noise filter.
Parameters	<ul style="list-style-type: none"> • hi2c: : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral. • DigitalFilter: : Coefficient of digital noise filter between 0x00 and 0x0F.
Return values	<ul style="list-style-type: none"> • HAL status

21.1.6 HAL_I2CEEx_EnableWakeUp

Function Name	<code>HAL_StatusTypeDef HAL_I2CEEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)</code>
Function Description	Enables I2C wakeup from stop mode.
Parameters	<ul style="list-style-type: none"> • hi2c: : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	<ul style="list-style-type: none"> • HAL status

21.1.7 HAL_I2CEEx_DisableWakeUp

Function Name	<code>HAL_StatusTypeDef HAL_I2CEEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)</code>
Function Description	Disables I2C wakeup from stop mode.
Parameters	<ul style="list-style-type: none"> • hi2c: : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	<ul style="list-style-type: none"> • HAL status

21.1.8 HAL_I2CEEx_EnableFastModePlus

Function Name	<code>void HAL_I2CEEx_EnableFastModePlus (uint32_t ConfigFastModePlus)</code>
Function Description	Enable the I2C fast mode plus driving capability.
Parameters	<ul style="list-style-type: none"> • ConfigFastModePlus: selects the pin. This parameter can be one of the I2CEEx Fast Mode Plus values
Return values	<ul style="list-style-type: none"> • None

21.1.9 HAL_I2CEEx_DisableFastModePlus

Function Name	void HAL_I2CEEx_DisableFastModePlus (uint32_t ConfigFastModePlus)
Function Description	Disable the I2C fast mode plus driving capability.
Parameters	<ul style="list-style-type: none">• ConfigFastModePlus: selects the pin. This parameter can be one of the I2CEEx Fast Mode Plus values
Return values	<ul style="list-style-type: none">• None

21.2 I2CEEx Firmware driver defines

21.2.1 I2CEEx

I2CEEx Analog Filter

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2CEEx Fast Mode Plus

I2C_FASTMODEPLUS_PA9	Enable Fast Mode Plus on PA9
I2C_FASTMODEPLUS_PA10	Enable Fast Mode Plus on PA10
I2C_FASTMODEPLUS_PB6	Enable Fast Mode Plus on PB6
I2C_FASTMODEPLUS_PB7	Enable Fast Mode Plus on PB7
I2C_FASTMODEPLUS_PB8	Enable Fast Mode Plus on PB8
I2C_FASTMODEPLUS_PB9	Enable Fast Mode Plus on PB9
I2C_FASTMODEPLUS_I2C1	Enable Fast Mode Plus on I2C1 pins
I2C_FASTMODEPLUS_I2C2	Enable Fast Mode Plus on I2C2 pins

22 HAL I2S Generic Driver

22.1 I2S Firmware driver registers structures

22.1.1 I2S_InitTypeDef

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

Data Fields

- *SPI_TypeDef * Instance*
- *I2S_InitTypeDef Init*
- *uint16_t * pTxBuffPtr*
- *__IO uint16_t TxXferSize*
- *__IO uint16_t TxXferCount*
- *uint16_t * pRxBuffPtr*
- *__IO uint16_t RxXferSize*
- *__IO uint16_t RxXferCount*
- *DMA_HandleTypeDef * hdmatx*

- **DMA_HandleTypeDef * hdmarx**
- **IO HAL_LockTypeDef Lock**
- **IO HAL_I2S_StateTypeDef State**
- **IO uint32_t ErrorCode**

Field Documentation

- **SPI_HandleTypeDef* I2S_HandleTypeDef::Instance**
I2S registers base address
- **I2S_InitTypeDef I2S_HandleTypeDef::Init**
I2S communication parameters
- **uint16_t* I2S_HandleTypeDef::pTxBuffPtr**
Pointer to I2S Tx transfer buffer
- **IO uint16_t I2S_HandleTypeDef::TxXferSize**
I2S Tx transfer size
- **IO uint16_t I2S_HandleTypeDef::TxXferCount**
I2S Tx transfer Counter
- **uint16_t* I2S_HandleTypeDef::pRxBuffPtr**
Pointer to I2S Rx transfer buffer
- **IO uint16_t I2S_HandleTypeDef::RxXferSize**
I2S Rx transfer size
- **IO uint16_t I2S_HandleTypeDef::RxXferCount**
I2S Rx transfer counter (This field is initialized at the same value as transfer size at the beginning of the transfer and decremented when a sample is received.
NbSamplesReceived = RxBufferSize-RxBufferCount)
- **DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx**
I2S Tx DMA handle parameters
- **DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx**
I2S Rx DMA handle parameters
- **IO HAL_LockTypeDef I2S_HandleTypeDef::Lock**
I2S locking object
- **IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State**
I2S communication state
- **IO uint32_t I2S_HandleTypeDef::ErrorCode**
I2S Error code This parameter can be a value of [I2S_Error](#)

22.2 I2S Firmware driver API description

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 I2S HAL driver.

- __HAL_I2S_ENABLE: Enable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_DISABLE: Disable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_ENABLE_IT : Enable the specified I2S interrupts
- __HAL_I2S_DISABLE_IT : Disable the specified I2S interrupts
- __HAL_I2S_GET_FLAG: Check whether the specified I2S flag is set or not



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.

This section contains the following APIs:

- [**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()

This section contains the following APIs:

- [*HAL_I2S_Transmit\(\)*](#)
- [*HAL_I2S_Receive\(\)*](#)
- [*HAL_I2S_Transmit_IT\(\)*](#)
- [*HAL_I2S_Receive_IT\(\)*](#)
- [*HAL_I2S_Transmit_DMA\(\)*](#)
- [*HAL_I2S_Receive_DMA\(\)*](#)
- [*HAL_I2S_DMAPause\(\)*](#)
- [*HAL_I2S_DMAResume\(\)*](#)
- [*HAL_I2S_DMAStop\(\)*](#)
- [*HAL_I2S_IRQHandler\(\)*](#)
- [*HAL_I2S_TxHalfCpltCallback\(\)*](#)
- [*HAL_I2S_TxCpltCallback\(\)*](#)
- [*HAL_I2S_RxHalfCpltCallback\(\)*](#)
- [*HAL_I2S_RxCpltCallback\(\)*](#)
- [*HAL_I2S_ErrorCallback\(\)*](#)

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.

This section contains the following APIs:

- [*HAL_I2S_GetState\(\)*](#)
- [*HAL_I2S_GetError\(\)*](#)

22.2.5 HAL_I2S_Init

Function Name **HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)**

Function Description	Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> HAL status
22.2.6 HAL_I2S_DelInit	
Function Name	HAL_StatusTypeDef HAL_I2S_DelInit (I2S_HandleTypeDef * hi2s)
Function Description	DeInitializes the I2S peripheral.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> HAL status
22.2.7 HAL_I2S_MspInit	
Function Name	void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)
Function Description	I2S MSP Init.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> None
22.2.8 HAL_I2S_MspDelInit	
Function Name	void HAL_I2S_MspDelInit (I2S_HandleTypeDef * hi2s)
Function Description	I2S MSP DelInit.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> None
22.2.9 HAL_I2S_Transmit	
Function Name	HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module pData: a 16-bit pointer to data buffer. Size: number of data sample to be sent: Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data

frame is selected the Size parameter means the number of 16-bit data length.

- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

22.2.10 HAL_I2S_Receive

Function Name	HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module pData: a 16-bit pointer to data buffer. Size: number of data sample to be sent: Timeout: Timeout duration
Return values	• HAL status
Notes	<ul style="list-style-type: none"> When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming). In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continuous way and as the I2S is not disabled at the end of the I2S transaction.

22.2.11 HAL_I2S_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_I2S_Transmit_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module pData: a 16-bit pointer to data buffer. Size: number of data sample to be sent:
Return values	• HAL status
Notes	<ul style="list-style-type: none"> When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

22.2.12 HAL_I2S_Receive_IT

Function Name	HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Receive an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Receive data buffer. • Size: number of data sample to be sent:
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming). • It is recommended to use DMA for the I2S receiver to avoid de-synchronisation between Master and Slave otherwise the I2S interrupt should be optimized.

22.2.13 HAL_I2S_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Transmit data buffer. • Size: number of data sample to be sent:
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

22.2.14 HAL_I2S_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_I2S_Receive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Receive an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

- **pData:** a 16-bit pointer to the Receive data buffer.
 - **Size:** number of data sample to be sent:
- | | |
|---------------|---|
| Return values | • HAL status |
| Notes | <ul style="list-style-type: none"> • When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming). |

22.2.15 HAL_I2S_DMAPause

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

22.2.16 HAL_I2S_DMAResume

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

22.2.17 HAL_I2S_DMAStop

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

22.2.18 HAL_I2S_IRQHandler

Function Name	void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)
Function Description	This function handles I2S interrupt request.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• None

22.2.19 HAL_I2S_TxHalfCpltCallback

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

22.2.20 HAL_I2S_TxCpltCallback

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

22.2.21 HAL_I2S_RxHalfCpltCallback

Function Name	void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Rx Transfer half completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None

22.2.22 HAL_I2S_RxCpltCallback

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

22.2.23 HAL_I2S_ErrorCallback

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

22.2.24 HAL_I2S_GetState

Function Name	HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * hi2s)
Function Description	Return the I2S state.

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

22.2.25 HAL_I2S_GetError

Function Name	uint32_t HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)
Function Description	Return the I2S error code.
Parameters	<ul style="list-style-type: none">• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none">• I2S Error Code

22.3 I2S Firmware driver defines

22.3.1 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 Error

HAL_I2S_ERROR_NONE No error

HAL_I2S_ERROR_TIMEOUT	Timeout error
HAL_I2S_ERROR_OVR	OVR error
HAL_I2S_ERROR_UDR	UDR error
HAL_I2S_ERROR_DMA	DMA transfer error
HAL_I2S_ERROR_UNKNOW	Unknow Error error

I2S Exported Macros

<code>_HAL_I2S_RESET_HANDLE_STATE</code>	Description: • Reset I2S handle state. Parameters: <ul style="list-style-type: none">• <code>_HANDLE_</code>: I2S handle. Return value: <ul style="list-style-type: none">• None
<code>_HAL_I2S_ENABLE</code>	Description: • Enable or disable the specified SPI peripheral (in I2S mode). Parameters: <ul style="list-style-type: none">• <code>_HANDLE_</code>: specifies the I2S Handle. Return value: <ul style="list-style-type: none">• None
<code>_HAL_I2S_DISABLE</code>	Description: • Enable or disable the specified I2S interrupts. Parameters: <ul style="list-style-type: none">• <code>_HANDLE_</code>: specifies the I2S Handle.• <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">– <code>I2S_IT_TXE</code>: Tx buffer empty interrupt enable– <code>I2S_IT_RXNE</code>: RX buffer not empty interrupt enable– <code>I2S_IT_ERR</code>: Error interrupt enable Return value: <ul style="list-style-type: none">• None
<code>_HAL_I2S_DISABLE_IT</code>	Description: • Checks if the specified I2S interrupt source is enabled or disabled. Parameters:
<code>_HAL_I2S_GET_IT_SOURCE</code>	

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2, or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the I2S interrupt source to check. This parameter can be one of the following values:
 - I2S_IT_TXE: Tx buffer empty interrupt enable
 - I2S_IT_RXNE: RX buffer not empty interrupt enable
 - I2S_IT_ERR: Error interrupt enable

Return value:

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

[__HAL_I2S_GET_FLAG](#)**Description:**

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

Parameters:

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

Return value:

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

[__HAL_I2S_CLEAR_OVRFLAG](#)**Description:**

- Clears the I2S OVR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

[__HAL_I2S_CLEAR_UDRFLAG](#)**Description:**

- Clears the I2S UDR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

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

Data Fields

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

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate***
This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hirda->Init.BaudRate)))
- ***uint32_t IRDA_InitTypeDef::WordLength***
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [*IRDAEx_Word_Length*](#)
- ***uint32_t IRDA_InitTypeDef::Parity***
Specifies the parity mode. This parameter can be a value of [*IRDA_Parity*](#)
Note:When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t IRDA_InitTypeDef::Mode***
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [*IRDA_Mode*](#)
- ***uint8_t IRDA_InitTypeDef::Prescaler***
Specifies the Prescaler value 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

Data Fields

- *USART_TypeDef * Instance*
- *IRDA_InitTypeDef Init*
- *uint8_t * pTxBuffPtr*
- *uint16_t TxXferSize*

- *uint16_t TxXferCount*
- *uint8_t * pRxBuffPtr*
- *uint16_t RxXferSize*
- *uint16_t RxXferCount*
- *uint16_t Mask*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *__IO HAL_IRDA_StateTypeDef State*
- *__IO uint32_t ErrorCode*

Field Documentation

- ***USART_TypeDef* IRDA_HandleTypeDef::Instance***
USART registers base address
- ***IRDA_InitTypeDef IRDA_HandleTypeDef::Init***
IRDA communication parameters
- ***uint8_t* IRDA_HandleTypeDef::pTxBuffPtr***
Pointer to IRDA Tx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::TxXferSize***
IRDA Tx Transfer size
- ***uint16_t IRDA_HandleTypeDef::TxXferCount***
IRDA Tx Transfer Counter
- ***uint8_t* IRDA_HandleTypeDef::pRxBuffPtr***
Pointer to IRDA Rx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::RxXferSize***
IRDA Rx Transfer size
- ***uint16_t IRDA_HandleTypeDef::RxXferCount***
IRDA Rx Transfer Counter
- ***uint16_t IRDA_HandleTypeDef::Mask***
USART RX RDR register mask
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmatx***
IRDA Tx DMA Handle parameters
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmarx***
IRDA Rx DMA Handle parameters
- ***HAL_LockTypeDef IRDA_HandleTypeDef::Lock***
Locking object
- ***__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::State***
IRDA communication state
- ***__IO uint32_t IRDA_HandleTypeDef::ErrorCode***
IRDA Error code This parameter can be a value of [IRDA_Error](#)

23.2 IRDA Firmware driver API description

23.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA_HandleTypeDef handle structure (eg. IRDA_HandleTypeDef hirda).
2. Initialize the IRDA low level resources by implementing the HAL_IRDA_MspInit() API in setting the associated USART or UART in IRDA mode:
 - Enable the USARTx/UARTx interface clock.

- USARTx/UARTx pins configuration:
 - Enable the clock for the USARTx/UARTx GPIOs.
 - Configure these USARTx/UARTx pins (TX as alternate function pull-up, RX as alternate function Input).
 - NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
 - Configure the USARTx/UARTx interrupt priority.
 - Enable the NVIC USARTx/UARTx IRQ handle.
 - 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.
 - 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 DMA 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 handle 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 half of transfer HAL_IRDA_TxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxHalfCpltCallback()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_IRDA_Receive_DMA()
- At reception half of transfer HAL_IRDA_RxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxHalfCpltCallback()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

IRDA HAL driver macros list

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

- __HAL_IRDA_ENABLE: Enable the IRDA peripheral
- __HAL_IRDA_DISABLE: Disable the IRDA peripheral
- __HAL_IRDA_GET_FLAG : Check whether the specified IRDA flag is set or not
- __HAL_IRDA_CLEAR_FLAG : Clear the specified IRDA pending flag
- __HAL_IRDA_ENABLE_IT: Enable the specified IRDA interrupt
- __HAL_IRDA_DISABLE_IT: Disable the specified IRDA interrupt
- __HAL_IRDA_GET_IT_SOURCE: Check whether or not the specified IRDA interrupt is enabled



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 asynchronous IRDA mode.

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. According to device capability (support or not of 7-bit word length), frame length is either defined by the M bit (8-bits or 9-bits) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit). 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() API follows the USART asynchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [**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.
 - No-Blocking mode: the communication is performed using Interrupts or DMA, these API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
 - [**HAL_IRDA_Transmit\(\)**](#)
 - [**HAL_IRDA_Receive\(\)**](#)
3. Non Blocking mode APIs with Interrupt are :
 - [**HAL_IRDA_Transmit_IT\(\)**](#)
 - [**HAL_IRDA_Receive_IT\(\)**](#)
 - [**HAL_IRDA_IRQHandler\(\)**](#)
4. Non Blocking mode functions with DMA are :
 - [**HAL_IRDA_Transmit_DMA\(\)**](#)
 - [**HAL_IRDA_Receive_DMA\(\)**](#)
 - [**HAL_IRDA_DMAPause\(\)**](#)
 - [**HAL_IRDA_DMAResume\(\)**](#)
 - [**HAL_IRDA_DMAStop\(\)**](#)
5. A set of Transfer Complete Callbacks are provided in Non Blocking mode:

- HAL_IRDA_TxHalfCpltCallback()
- HAL_IRDA_TxCpltCallback()
- HAL_IRDA_RxHalfCpltCallback()
- HAL_IRDA_RxCpltCallback()
- HAL_IRDA_ErrorCallback()

This section contains the following APIs:

- [*HAL_IRDA_Transmit\(\)*](#)
- [*HAL_IRDA_Receive\(\)*](#)
- [*HAL_IRDA_Transmit_IT\(\)*](#)
- [*HAL_IRDA_Receive_IT\(\)*](#)
- [*HAL_IRDA_Transmit_DMA\(\)*](#)
- [*HAL_IRDA_Receive_DMA\(\)*](#)
- [*HAL_IRDA_DMAPause\(\)*](#)
- [*HAL_IRDA_DMAResume\(\)*](#)
- [*HAL_IRDA_DMAStop\(\)*](#)
- [*HAL_IRDA_IRQHandler\(\)*](#)
- [*HAL_IRDA_TxCpltCallback\(\)*](#)
- [*HAL_IRDA_TxHalfCpltCallback\(\)*](#)
- [*HAL_IRDA_RxCpltCallback\(\)*](#)
- [*HAL_IRDA_RxHalfCpltCallback\(\)*](#)
- [*HAL_IRDA_ErrorCallback\(\)*](#)

23.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process

- HAL_IRDA_GetState() API can be helpful to check in run-time the state of the IRDA peripheral handle.
- HAL_IRDA_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- [*HAL_IRDA_GetState\(\)*](#)
- [*HAL_IRDA_GetError\(\)*](#)

23.2.5 HAL_IRDA_Init

Function Name	HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)
Function Description	Initialize the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and initialize the associated handle.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • HAL status

23.2.6 HAL_IRDA_DeInit

Function Name	HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)
---------------	---

Function Description	Deinitialize the IRDA peripheral.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> HAL status

23.2.7 HAL_IRDA_MspInit

Function Name	void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)
Function Description	Initialize the IRDA MSP.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> None

23.2.8 HAL_IRDA_MspDeInit

Function Name	void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)
Function Description	Deinitialize the IRDA MSP.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> None

23.2.9 HAL_IRDA_Transmit

Function Name	HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. pData: Pointer to data buffer. Size: Amount of data to be sent. Timeout: Specify timeout value.
Return values	<ul style="list-style-type: none"> HAL status

23.2.10 HAL_IRDA_Receive

Function Name	HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. pData: Pointer to data buffer.

- **Size:** Amount of data to be received.
- **Timeout:** Specify timeout value.
- HAL status

Return values

23.2.11 HAL_IRDA_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. • pData: Pointer to data buffer. • Size: Amount of data to be sent.
Return values	• HAL status

23.2.12 HAL_IRDA_Receive_IT

Function Name	HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. • pData: Pointer to data buffer. • Size: Amount of data to be received.
Return values	• HAL status

23.2.13 HAL_IRDA_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module. • pData: pointer to data buffer. • Size: amount of data to be sent.
Return values	• HAL status

23.2.14 HAL_IRDA_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

- **pData:** Pointer to data buffer.
 - **Size:** Amount of data to be received.
- Return values
- HAL status
- Notes
- When the IRDA parity is enabled (PCE = 1) the received data contains the parity bit (MSB position).

23.2.15 HAL_IRDA_DMAPause

Function Name	HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)
Function Description	Pause the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • HAL status

23.2.16 HAL_IRDA_DMAResume

Function Name	HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)
Function Description	Resume the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> • HAL status

23.2.17 HAL_IRDA_DMAStop

Function Name	HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)
Function Description	Stop the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> • HAL status

23.2.18 HAL_IRDA_IRQHandler

Function Name	void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)
Function Description	Handle IRDA interrupt request.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • None

23.2.19 HAL_IRDA_TxCpltCallback

Function Name	<code>void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)</code>
Function Description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • None

23.2.20 HAL_IRDA_TxHalfCpltCallback

Function Name	<code>void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)</code>
Function Description	Tx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none"> • None

23.2.21 HAL_IRDA_RxCpltCallback

Function Name	<code>void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)</code>
Function Description	Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • None

23.2.22 HAL_IRDA_RxHalfCpltCallback

Function Name	<code>void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)</code>
Function Description	Rx Half Transfer complete callback.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • None

23.2.23 HAL_IRDA_ErrorCallback

Function Name	<code>void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)</code>
Function Description	IRDA error callback.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA

module.

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

23.2.24 HAL_IRDA_GetState

Function Name	HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)
Function Description	Return the IRDA handle state.
Parameters	<ul style="list-style-type: none">hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none">HAL state

23.2.25 HAL_IRDA_GetError

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

23.3 IRDA Firmware driver defines

23.3.1 IRDA

IRDA DMA Rx

IRDA_DMA_RX_DISABLE	IRDA DMA RX disabled
IRDA_DMA_RX_ENABLE	IRDA DMA RX enabled

IRDA DMA Tx

IRDA_DMA_TX_DISABLE	IRDA DMA TX disabled
IRDA_DMA_TX_ENABLE	IRDA DMA TX enabled

IRDA Error

HAL_IRDA_ERROR_NONE	No error
HAL_IRDA_ERROR_PE	Parity error
HAL_IRDA_ERROR_NE	Noise error
HAL_IRDA_ERROR_FE	frame error
HAL_IRDA_ERROR_ORE	Overrun error
HAL_IRDA_ERROR_DMA	DMA transfer error

IRDA Exported Macros

<u>_HAL_IRDA_RESET_HANDLE_STATE</u>	Description:
-------------------------------------	---------------------

- Reset IRDA handle state.

Parameters:

- `__HANDLE__`: IRDA handle.

Return value:

- None

`__HAL_IRDA_FLUSH_DRREGISTER`

Description:

- Flush the IRDA DR register.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

`__HAL_IRDA_CLEAR_FLAG`

Description:

- Clear the specified IRDA pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.
- `__FLAG__`: specifies the flag to check.
This parameter can be any combination of the following values:
 - `IRDA_CLEAR_PEF`
 - `IRDA_CLEAR_FEF`
 - `IRDA_CLEAR_NEF`
 - `IRDA_CLEAR_OREF`
 - `IRDA_CLEAR_TCF`
 - `IRDA_CLEAR_IDLEF`

Return value:

- None

`__HAL_IRDA_CLEAR_PEF`

Description:

- Clear the IRDA PE pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

`__HAL_IRDA_CLEAR_FEF`

Description:

- Clear the IRDA FE pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

`_HAL_IRDA_CLEAR_NEFLAG`

Description:

- Clear the IRDA NE pending flag.

Parameters:

- `_HANDLE_`: specifies the IRDA Handle.

Return value:

- None

`_HAL_IRDA_CLEAR_OREFLAG`

Description:

- Clear the IRDA ORE pending flag.

Parameters:

- `_HANDLE_`: specifies the IRDA Handle.

Return value:

- None

`_HAL_IRDA_CLEAR_IDLEFLAG`

Description:

- Clear the IRDA IDLE pending flag.

Parameters:

- `_HANDLE_`: specifies the IRDA Handle.

Return value:

- None

`_HAL_IRDA_GET_FLAG`

Description:

- Check 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_TXACK: 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:**

- Enable 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**Description:**

- Disable 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 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](#)

Description:

- Check whether the specified IRDA interrupt has occurred 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
- [__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:

- Check whether the specified IRDA interrupt source is enabled 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

- __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:

- Clear 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

Description:

- 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_ONE_BIT_SAMPLE_ENA
BLE`

Description:

- Enable the IRDA one bit sample method.

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_ONE_BIT_SAMPLE_DISA
BLE`

Description:

- Disable the IRDA one bit sample method.

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_ENABLE`

Description:

- Enable USART/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

<code>__HAL_IRDA_DISABLE</code>	<p>Description:</p> <ul style="list-style-type: none"> Disable UART/USART associated to IRDA Handle. <p>Parameters:</p> <ul style="list-style-type: none"> <code>__HANDLE__</code>: specifies the IRDA Handle. The Handle Instance can be USARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral <p>Return value:</p> <ul style="list-style-type: none"> None
---------------------------------	---

IRDA Flags

<code>IRDA_FLAG_RXACK</code>	IRDA Receive enable acknowledge flag
<code>IRDA_FLAG_TEACK</code>	IRDA Transmit enable acknowledge flag
<code>IRDA_FLAG_BUSY</code>	IRDA Busy flag
<code>IRDA_FLAG_ABRF</code>	IRDA Auto baud rate flag
<code>IRDA_FLAG_ABRE</code>	IRDA Auto baud rate error
<code>IRDA_FLAG_TXE</code>	IRDA Transmit data register empty
<code>IRDA_FLAG_TC</code>	IRDA Transmission complete
<code>IRDA_FLAG_RXNE</code>	IRDA Read data register not empty
<code>IRDA_FLAG_ORE</code>	IRDA Overrun error
<code>IRDA_FLAG_NE</code>	IRDA Noise error
<code>IRDA_FLAG_FE</code>	IRDA Noise error
<code>IRDA_FLAG_PE</code>	IRDA Parity error

IRDA interruptions flags mask

<code>IRDA_IT_MASK</code>	IRDA Interruptions flags mask
---------------------------	-------------------------------

IRDA Interrupts Definition

<code>IRDA_IT_PE</code>	IRDA Parity error interruption
<code>IRDA_IT_TXE</code>	IRDA Transmit data register empty interruption
<code>IRDA_IT_TC</code>	IRDA Transmission complete interruption
<code>IRDA_IT_RXNE</code>	IRDA Read data register not empty interruption
<code>IRDA_IT_IDLE</code>	IRDA Idle interruption
<code>IRDA_IT_ERR</code>	
<code>IRDA_IT_ORE</code>	
<code>IRDA_IT_NE</code>	IRDA Noise error interruption
<code>IRDA_IT_FE</code>	IRDA Frame error interruption

IRDA Interruption Clear Flags

<code>IRDA_CLEAR_PEF</code>	Parity Error Clear Flag
<code>IRDA_CLEAR_FEF</code>	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 normal power mode
IRDA_POWERMODE_LOWPOWER	IRDA low power mode

IRDA Mode

IRDA_MODE_DISABLE	Associated UART disabled in IRDA mode
IRDA_MODE_ENABLE	Associated UART enabled in IRDA mode

IRDA One Bit Sampling

IRDA_ONE_BIT_SAMPLE_DISABLE	One-bit sampling disabled
IRDA_ONE_BIT_SAMPLE_ENABLE	One-bit sampling enabled

IRDA Parity

IRDA_PARITY_NONE	No parity
IRDA_PARITY EVEN	Even parity
IRDA_PARITY ODD	Odd parity

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

IRDA State

IRDA_STATE_DISABLE	IRDA disabled
IRDA_STATE_ENABLE	IRDA enabled

IRDA Transfer Mode

IRDA_MODE_RX	RX mode
IRDA_MODE_TX	TX mode
IRDA_MODE_TX_RX	RX and TX mode

24 HAL IRDA Extension Driver

24.1 IRDAEx Firmware driver defines

24.1.1 IRDAEx

IRDA Word Length

IRDA_WORDLENGTH_7B 7-bit long frame

IRDA_WORDLENGTH_8B 8-bit long frame

IRDA_WORDLENGTH_9B 9-bit long frame

25 HAL IWDG Generic Driver

25.1 IWDG Firmware driver registers structures

25.1.1 IWDG_InitTypeDef

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

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

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
- `IWDG_ENABLE_WRITE_ACCESS` : Enable write access to IWDG_PR and IWDG_RLR registers
- `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 and de-initialization functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the `IWDG_InitTypeDef` and create the associated handle
- Manage Window option
- Initialize the IWDG MSP
- Deinitialize the IWDG MSP

This section contains the following APIs:

- `HAL_IWDG_Init()`
- `HAL_IWDG_MspInit()`

25.2.4 IO operation functions

This section provides functions allowing to:

- Start the IWDG.
- Refresh the IWDG.

This section contains the following APIs:

- `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.

This section contains the following APIs:

- `HAL_IWDG_GetState()`

25.2.6 HAL_IWDG_Init

Function Name	<code>HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef *hiwdg)</code>
Function Description	Initialize the IWDG according to the specified parameters in the <code>IWDG_InitTypeDef</code> and initialize the associated handle.
Parameters	<ul style="list-style-type: none">• hiwdg: pointer to a <code>IWDG_HandleTypeDef</code> structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none">• HAL status

25.2.7 HAL_IWDG_MspInit

Function Name	void HAL_IWDG_MspInit (IWDG_HandleTypeDef * hiwdg)
Function Description	Initialize the IWDG MSP.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • None

25.2.8 HAL_IWDG_Start

Function Name	HAL_StatusTypeDef HAL_IWDG_Start (IWDG_HandleTypeDef * hiwdg)
Function Description	Start the IWDG.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

25.2.9 HAL_IWDG_Refresh

Function Name	HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)
Function Description	Refresh the IWDG.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

25.2.10 HAL_IWDG_GetState

Function Name	HAL_IWDG_StateTypeDef HAL_IWDG_GetState (IWDG_HandleTypeDef * hiwdg)
Function Description	Return the IWDG handle state.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • HAL state

25.3 IWDG Firmware driver defines

25.3.1 IWDG

IWDG Exported Macros

`_HAL_IWDG_RESET_HANDLE_STATE` **Description:**

- Reset IWDG handle state.

Parameters:

- `__HANDLE__`: IWDG handle.

Return value:

- None

`__HAL_IWDG_START`**Description:**

- Enable the IWDG peripheral.

Parameters:

- `__HANDLE__`: IWDG handle

Return value:

- None

`__HAL_IWDG_RELOAD_COUNTER`**Description:**

- Reload IWDG counter with value defined in the reload register.

Parameters:

- `__HANDLE__`: IWDG handle

Return value:

- None

`__HAL_IWDG_GET_FLAG`**Description:**

- Get the selected IWDG 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 Prescaler

<code>IWDG_PRESCALER_4</code>	IWDG prescaler set to 4
<code>IWDG_PRESCALER_8</code>	IWDG prescaler set to 8
<code>IWDG_PRESCALER_16</code>	IWDG prescaler set to 16
<code>IWDG_PRESCALER_32</code>	IWDG prescaler set to 32
<code>IWDG_PRESCALER_64</code>	IWDG prescaler set to 64
<code>IWDG_PRESCALER_128</code>	IWDG prescaler set to 128

IWDG_PRESCALER_256 IWDG prescaler set to 256

IWDG Window

IWDG_WINDOW_DISABLE

26 HAL PCD Generic Driver

26.1 PCD Firmware driver registers structures

26.1.1 PCD_InitTypeDef

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

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_HandleTypeDef

Data Fields



- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***_IO uint8_t USB_Address***
- ***PCD_EPTTypeDef IN_ep***
- ***PCD_EPTTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***_IO PCD_StateTypeDef State***
- ***uint32_t Setup***
- ***void * pData***

Field Documentation

- ***PCD_TypeDef* PCD_HandleTypeDef::Instance***
Register base address
- ***PCD_InitTypeDef PCD_HandleTypeDef::Init***
PCD required parameters
- ***_IO uint8_t PCD_HandleTypeDef::USB_Address***
USB Address
- ***PCD_EPTTypeDef PCD_HandleTypeDef::IN_ep[8]***
IN endpoint parameters
- ***PCD_EPTTypeDef PCD_HandleTypeDef::OUT_ep[8]***
OUT endpoint parameters
- ***HAL_LockTypeDef PCD_HandleTypeDef::Lock***
PCD peripheral status
- ***_IO PCD_StateTypeDef PCD_HandleTypeDef::State***
PCD communication state
- ***uint32_t PCD_HandleTypeDef::Setup[12]***
Setup packet buffer
- ***void* PCD_HandleTypeDef::pData***
Pointer to upper stack Handler

26.2 PCD Firmware driver API description

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
 - __HAL_RCC_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:

This section contains the following APIs:

- *HAL_PCD_Init()*
- *HAL_PCD_DelInit()*
- *HAL_PCD_MspInit()*
- *HAL_PCD_MspDelInit()*

26.2.3 IO operation functions

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

This section contains the following APIs:

- *HAL_PCD_Start()*
- *HAL_PCD_Stop()*
- *HAL_PCD_IRQHandler()*
- *HAL_PCD_DataOutStageCallback()*
- *HAL_PCD_DataInStageCallback()*
- *HAL_PCD_SetupStageCallback()*
- *HAL_PCD_SOFCallback()*
- *HAL_PCD_ResetCallback()*
- *HAL_PCD_SuspendCallback()*
- *HAL_PCD_ResumeCallback()*
- *HAL_PCD_ISOOUTIncompleteCallback()*
- *HAL_PCD_ISOINIncompleteCallback()*
- *HAL_PCD_ConnectCallback()*
- *HAL_PCD_DisconnectCallback()*

26.2.4 Peripheral Control functions

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

This section contains the following APIs:

- *HAL_PCD_DevConnect()*
- *HAL_PCD_DevDisconnect()*
- *HAL_PCD_SetAddress()*
- *HAL_PCD_EP_Open()*
- *HAL_PCD_EP_Close()*
- *HAL_PCD_EP_Receive()*
- *HAL_PCD_EP_GetRxCount()*
- *HAL_PCD_EP_Transmit()*
- *HAL_PCD_EP_SetStall()*
- *HAL_PCD_EP_ClrStall()*
- *HAL_PCD_EP_Flush()*
- *HAL_PCD_ActivateRemoteWakeup()*
- *HAL_PCD_DeActivateRemoteWakeup()*

26.2.5 Peripheral State functions

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

This section contains the following APIs:

- *HAL_PCD_GetState()*

26.2.6 HAL_PCD_Init

Function Name	HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)
Function Description	Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• HAL status

26.2.7 HAL_PCD_DelInit

Function Name	HAL_StatusTypeDef HAL_PCD_DelInit (PCD_HandleTypeDef * hpcd)
Function Description	Deinitializes the PCD peripheral.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• HAL status

26.2.8 HAL_PCD_MspInit

Function Name	void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)
Function Description	Initializes the PCD MSP.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• None

26.2.9 HAL_PCD_MspDelInit

Function Name	void HAL_PCD_MspDelInit (PCD_HandleTypeDef * hpcd)
Function Description	Deinitializes PCD MSP.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• None

26.2.10 HAL_PCD_Start

Function Name	HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)
Function Description	Start the USB device.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• HAL status

26.2.11 HAL_PCD_Stop

Function Name	HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)
Function Description	Stop the USB device.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle

Return values • HAL status

26.2.12 HAL_PCD_IRQHandler

Function Name **void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)**

Function Description This function handles PCD interrupt request.

Parameters • **hpcd**: PCD handle

Return values • HAL status

26.2.13 HAL_PCD_DataOutStageCallback

Function Name **void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**

Function Description Data out stage callbacks.

Parameters • **hpcd**: PCD handle
• **epnum**: endpoint number

Return values • None

26.2.14 HAL_PCD_DataInStageCallback

Function Name **void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**

Function Description Data IN stage callbacks.

Parameters • **hpcd**: PCD handle
• **epnum**: endpoint number

Return values • None

26.2.15 HAL_PCD_SetupStageCallback

Function Name **void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)**

Function Description Setup stage callback.

Parameters • **hpcd**: PCD handle

Return values • None

26.2.16 HAL_PCD_SOFCallback

Function Name **void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)**

Function Description USB Start Of Frame callbacks.

Parameters • **hpcd**: PCD handle

Return values • None

26.2.17 HAL_PCD_ResetCallback

Function Name **void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)**

Function Description	USB Reset callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • None

26.2.18 HAL_PCD_SuspendCallback

Function Name	void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)
Function Description	Suspend event callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • None

26.2.19 HAL_PCD_ResumeCallback

Function Name	void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)
Function Description	Resume event callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • None

26.2.20 HAL_PCD_ISOOUTIncompleteCallback

Function Name	void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)
Function Description	Incomplete ISO OUT callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • epnum: endpoint number
Return values	<ul style="list-style-type: none"> • None

26.2.21 HAL_PCD_ISOINIncompleteCallback

Function Name	void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)
Function Description	Incomplete ISO IN callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • epnum: endpoint number
Return values	<ul style="list-style-type: none"> • None

26.2.22 HAL_PCD_ConnectCallback

Function Name	void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)
Function Description	Connection event callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • None

26.2.23 HAL_PCD_DisconnectCallback

Function Name	<code>void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)</code>
Function Description	Disconnection event callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • None

26.2.24 HAL_PCD_DevConnect

Function Name	<code>HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)</code>
Function Description	Connect the USB device.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • HAL status

26.2.25 HAL_PCD_DevDisconnect

Function Name	<code>HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)</code>
Function Description	Disconnect the USB device.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • HAL status

26.2.26 HAL_PCD_SetAddress

Function Name	<code>HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)</code>
Function Description	Set the USB Device address.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • address: new device address
Return values	<ul style="list-style-type: none"> • HAL status

26.2.27 HAL_PCD_EP_Open

Function Name	<code>HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)</code>
Function Description	Open and configure an endpoint.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address • ep_mps: endpoint max packet size • ep_type: endpoint type
Return values	<ul style="list-style-type: none"> • HAL status

26.2.28 HAL_PCD_EP_Close

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function Description	Deactivate an endpoint.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address
Return values	<ul style="list-style-type: none"> • HAL status

26.2.29 HAL_PCD_EP_Receive

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)
Function Description	Receive an amount of data.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address • pBuf: pointer to the reception buffer • len: amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status

26.2.30 HAL_PCD_EP_GetRxCount

Function Name	uint16_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function Description	Get Received Data Size.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address
Return values	<ul style="list-style-type: none"> • Data Size

26.2.31 HAL_PCD_EP_Transmit

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)
Function Description	Send an amount of data.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address • pBuf: pointer to the transmission buffer • len: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

26.2.32 HAL_PCD_EP_SetStall

Function Name	HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function Description	Set a STALL condition over an endpoint.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle

- **ep_addr:** endpoint address

Return values • HAL status

26.2.33 HAL_PCD_EP_ClrStall

Function Name	HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function Description	Clear a STALL condition over in an endpoint.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address
Return values	• HAL status

26.2.34 HAL_PCD_EP_Flush

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function Description	Flush an endpoint.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address
Return values	• HAL status

26.2.35 HAL_PCD_ActivateRemoteWakeup

Function Name	HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)
Function Description	HAL_PCD_ActivateRemoteWakeup : active remote wakeup signalling.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	• HAL status

26.2.36 HAL_PCD_DeActivateRemoteWakeup

Function Name	HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)
Function Description	HAL_PCD_DeActivateRemoteWakeup : de-active remote wakeup signalling.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	• HAL status

26.2.37 HAL_PCD_GetState

Function Name	PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)
Function Description	Return the PCD state.

Parameters	• hpcd: : PCD handle
Return values	• HAL state

26.3 PCD Firmware driver defines

26.3.1 PCD

PCD Core PHY

PCD_PHY_EMBEDDED

PCD Core Speed

PCD_SPEED_HIGH

PCD_SPEED_FULL

PCD ENDP

PCD_ENDP0

PCD_ENDP1

PCD_ENDP2

PCD_ENDP3

PCD_ENDP4

PCD_ENDP5

PCD_ENDP6

PCD_ENDP7

PCD Endpoint Kind

PCD_SNG_BUF

PCD_DBL_BUF

PCD EP0 MPS

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

__HAL_PCD_CLEAR_FLAG

__HAL_USB_WAKEUP_EXTI_ENABLE_IT

__HAL_USB_WAKEUP_EXTI_DISABLE_IT

__HAL_USB_EXTI_GENERATE_SWIT

PCD Instance definition

IS_PCD_ALL_INSTANCE

27 HAL PCD Extension Driver

27.1 PCDEEx Firmware driver API description

27.1.1 Extended Peripheral Control functions

This section provides functions allowing to:

- Update PMA configuration

This section contains the following APIs:

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

27.1.2 HAL_PCDEEx_PMAConfig

Function Name	<code>HAL_StatusTypeDef HAL_PCDEEx_PMAConfig (PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t ep_kind, uint32_t pmaaddress)</code>
Function Description	Configure PMA for EP.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• ep_addr: endpoint address• ep_kind: endpoint Kind USB_SNG_BUF: Single Buffer usedUSB_DBL_BUF: Double Buffer used• pmaaddress: EP address in The PMA: In case of single buffer endpoint this parameter is 16-bit value providing the address in PMA allocated to endpoint. In case of double buffer endpoint this parameter is a 32-bit value providing the endpoint buffer 0 address in the LSB part of 32-bit value and endpoint buffer 1 address in the MSB part of 32-bit value.
Return values	<ul style="list-style-type: none">• : status

28 HAL PWR Generic Driver

28.1 PWR Firmware driver API description

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 `_HAL_RCC_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.

This section contains the following APIs:

- `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	<code>void HAL_PWR_EnableBkUpAccess (void)</code>
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers when present).
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

28.1.4 HAL_PWR_DisableBkUpAccess

Function Name	<code>void HAL_PWR_DisableBkUpAccess (void)</code>
Function Description	Disables access to the backup domain (RTC registers, RTC backup data registers when present).
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

28.1.5 HAL_PWR_EnableWakeUpPin

Function Name	<code>void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)</code>
Function Description	Enables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> • WakeUpPinx: Specifies the Power Wake-Up pin to enable. This parameter can be value of : PWREx Wakeup Pins
Return values	<ul style="list-style-type: none"> • None

28.1.6 HAL_PWR_DisableWakeUpPin

Function Name	<code>void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)</code>
Function Description	Disables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> • WakeUpPinx: Specifies the Power Wake-Up pin to disable.

This parameter can be values of : PWREx Wakeup Pins

Return values

- None

28.1.7 HAL_PWR_EnterSLEEPMode

Function Name

`void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)`

Function Description

Enters Sleep mode.

Parameters

- **Regulator:** Specifies the regulator state in SLEEP mode. 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.
- **SLEEPEntry:** Specifies if SLEEP mode is entered with WFI or WFE instruction. When WFI entry is used, tick interrupt have to be disabled if not desired as the interrupt wake up source. This parameter can be one of the following values: PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instructionPWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction

Return values

- None

Notes

- In Sleep mode, all I/O pins keep the same state as in Run mode.

28.1.8 HAL_PWR_EnterSTOPMode

Function Name

`void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)`

Function Description

Enters STOP mode.

Parameters

- **Regulator:** Specifies the regulator state in STOP mode. This parameter can be one of the following values: PWR_MAINREGULATOR_ON: STOP mode with regulator ONPWR_LOWPOWERREGULATOR_ON: STOP mode with low power regulator ON
- **STOPEntry:** specifies if STOP mode is entered with WFI or WFE instruction. This parameter can be one of the following values: PWR_STOPENTRY_WFI: Enter STOP mode with WFI instructionPWR_STOPENTRY_WFE: Enter STOP mode with WFE instruction

Return values

- None

Notes

- In Stop mode, all I/O pins keep the same state as in Run mode.
- When exiting Stop mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock.
- When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

28.1.9 HAL_PWR_EnterSTANDBYMode

Function Name	void HAL_PWR_EnterSTANDBYMode (void)
Function Description	Enters STANDBY mode.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • In Standby mode, all I/O pins are high impedance except for: Reset pad (still available)RTC 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.

28.1.10 HAL_PWR_EnableSleepOnExit

Function Name	void HAL_PWR_EnableSleepOnExit (void)
Function Description	Indicates Sleep-On-Exit when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • 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.

28.1.11 HAL_PWR_DisableSleepOnExit

Function Name	void HAL_PWR_DisableSleepOnExit (void)
Function Description	Disables Sleep-On-Exit feature when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

28.1.12 HAL_PWR_EnableSEVOnPend

Function Name	void HAL_PWR_EnableSEVOnPend (void)
Function Description	Enables CORTEX M4 SEVONPEND bit.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • 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.

28.1.13 HAL_PWR_DisableSEVOnPend

Function Name	void HAL_PWR_DisableSEVOnPend (void)
Function Description	Disables CORTEX M4 SEVONPEND bit.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • 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.

28.1.14 HAL_PWR_EnableBkUpAccess

Function Name	void HAL_PWR_EnableBkUpAccess (void)
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers when present).
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

28.1.15 HAL_PWR_DisableBkUpAccess

Function Name	void HAL_PWR_DisableBkUpAccess (void)
Function Description	Disables access to the backup domain (RTC registers, RTC backup data registers when present).
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

28.2 PWR Firmware driver defines

28.2.1 PWR

PWR Exported Macro

<u>__HAL_PWR_GET_FLAG</u>	Description:
	<ul style="list-style-type: none"> • Check PWR flag is set or not.
	Parameters:
	<ul style="list-style-type: none"> • <u>__FLAG__</u>: specifies the flag to check. This parameter can be one of the following values: <ul style="list-style-type: none"> – 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 STM32F030x8 products

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

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

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_ConfigPVD()`, `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_PWREx_EnableVddio2Monitor()` function. VDDIO2 is available on STM32F07x/09x/04x

This section contains the following APIs:

- [HAL_PWR_ConfigPVD\(\)](#)
- [HAL_PWR_EnablePVD\(\)](#)
- [HAL_PWR_DisablePVD\(\)](#)

- [***HAL_PWR_PVD_IRQHandler\(\)***](#)
- [***HAL_PWR_PVDCallback\(\)***](#)
- [***HAL_PWREx_EnableVddio2Monitor\(\)***](#)
- [***HAL_PWREx_DisableVddio2Monitor\(\)***](#)
- [***HAL_PWREx_Vddio2Monitor_IRQHandler\(\)***](#)
- [***HAL_PWREx_Vddio2MonitorCallback\(\)***](#)

29.2.2 HAL_PWR_ConfigPVD

Function Name	void HAL_PWR_ConfigPVD (PWR_PVDTTypeDef * sConfigPVD)
Function Description	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"> • sConfigPVD: pointer to an PWR_PVDTTypeDef structure that contains the configuration information for the PVD.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.

29.2.3 HAL_PWR_EnablePVD

Function Name	void HAL_PWR_EnablePVD (void)
Function Description	Enables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> • None

29.2.4 HAL_PWR_DisablePVD

Function Name	void HAL_PWR_DisablePVD (void)
Function Description	Disables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> • None

29.2.5 HAL_PWR_PVD_IRQHandler

Function Name	void HAL_PWR_PVD_IRQHandler (void)
Function Description	This function handles the PWR PVD interrupt request.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This API should be called under the PVD_IRQHandler() or PVD_VDDIO2_IRQHandler().

29.2.6 HAL_PWR_PVDCallback

Function Name	void HAL_PWR_PVDCallback (void)
Function Description	PWR PVD interrupt callback.
Return values	<ul style="list-style-type: none"> • None

29.2.7 HAL_PWREx_EnableVddio2Monitor

Function Name	void HAL_PWREx_EnableVddio2Monitor (void)
Function Description	Enable VDDIO2 monitor: enable Exti 31 and falling edge detection.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • If Exti 31 is enable correly and VDDIO2 voltage goes below Vrefint, an interrupt is generated Irq line 1. NVIS has to be enable by user.

29.2.8 **HAL_PWREx_DisableVddio2Monitor**

Function Name	void HAL_PWREx_DisableVddio2Monitor (void)
Function Description	Disable the Vddio2 Monitor.
Return values	<ul style="list-style-type: none"> • None

29.2.9 **HAL_PWREx_Vddio2Monitor_IRQHandler**

Function Name	void HAL_PWREx_Vddio2Monitor_IRQHandler (void)
Function Description	This function handles the PWR Vddio2 monitor interrupt request.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This API should be called under the VDDIO2_IRQHandler() PVD_VDDIO2_IRQHandler().

29.2.10 **HAL_PWREx_Vddio2MonitorCallback**

Function Name	void HAL_PWREx_Vddio2MonitorCallback (void)
Function Description	PWR Vddio2 Monitor interrupt callback.
Return values	<ul style="list-style-type: none"> • None

29.3 PWREx Firmware driver defines

29.3.1 PWREx

PWREx Exported Macros

<u>__HAL_PWR_PVD_EXTI_ENABLE_IT</u>	Description:
	<ul style="list-style-type: none"> • Enable interrupt on PVD Exti Line 16.
<u>__HAL_PWR_PVD_EXTI_DISABLE_IT</u>	Return value:
	<ul style="list-style-type: none"> • None.
<u>__HAL_PWR_PVD_EXTI_ENABLE_EVENT</u>	Description:
	<ul style="list-style-type: none"> • Disable interrupt on PVD Exti Line 16.
<u>__HAL_PWR_PVD_EXTI_DISABLE_EVENT</u>	Return value:
	<ul style="list-style-type: none"> • None.

Return value:

- None.

`__HAL_PWR_PVD_EXTI_DISABLE_EVENT`

Description:

- Disable event on PVD Exti Line 16.

Return value:

- None.

`__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE`

Description:

- Disable the PVD Extended Interrupt Rising Trigger.

Return value:

- None.

`__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE`

Description:

- Disable the PVD Extended Interrupt Falling Trigger.

Return value:

- None.

`__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE`

Description:

- Disable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None

`__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE`

Description:

- PVD EXTI line configuration: set falling edge trigger.

Return value:

- None.

`__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE`

Description:

- PVD EXTI line configuration: set rising edge trigger.

Return value:

- None.

`__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE`

Description:

- Enable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None

Description:

- Check whether the specified PVD EXTI interrupt flag is set or not.

Return value:

- EXTI: PVD Line Status.

Description:

- Clear the PVD EXTI flag.

Return value:

- None.

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None.

Description:

- Enable interrupt on Vddio2 Monitor Exti Line 31.

Return value:

- None.

Description:

- Disable interrupt on Vddio2 Monitor Exti Line 31.

Return value:

- None.

Description:

- Vddio2 Monitor EXTI line configuration: clear

falling edge and rising edge trigger.

Return value:

- None.

`_HAL_PWR_VDDIO2_EXTI_ENABLE_FALLING_EDGE`

Description:

- Vddio2 Monitor EXTI line configuration: set falling edge trigger.

Return value:

- None.

`_HAL_PWR_VDDIO2_EXTI_GET_FLAG`

Description:

- Check whether the specified VDDIO2 monitor EXTI interrupt flag is set or not.

Return value:

- EXTI: VDDIO2 Monitor Line Status.

`_HAL_PWR_VDDIO2_EXTI_CLEAR_FLAG`

Description:

- Clear the VDDIO2 Monitor EXTI flag.

Return value:

- None.

`_HAL_PWR_VDDIO2_EXTI_GENERATE_SWIT`

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None.

PWREx EXTI Line

`PWR_EXTI_LINE_PVD` External interrupt line 16 Connected to the PVD EXTI Line

`PWR_EXTI_LINE_VDDIO2` External interrupt line 31 Connected to the Vddio2 Monitor EXTI Line

PWREx Flag

`PWR_FLAG_WU`

`PWR_FLAG_SB`

`PWR_FLAG_PVDO`

`PWR_FLAG_VREFINTRDY`

PWREx PVD detection level

PWR_PVDLEVEL_0
PWR_PVDLEVEL_1
PWR_PVDLEVEL_2
PWR_PVDLEVEL_3
PWR_PVDLEVEL_4
PWR_PVDLEVEL_5
PWR_PVDLEVEL_6
PWR_PVDLEVEL_7
IS_PWR_PVD_LEVEL

PWREx PVD Mode

PWR_PVD_MODE_NORMAL	basic mode is used
PWR_PVD_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
PWR_PVD_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
PWR_PVD_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING	Event Mode with Rising edge trigger detection
PWR_PVD_MODE_EVENT_FALLING	Event Mode with Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING_FALLING	Event Mode with Rising/Falling edge trigger detection

IS_PWR_PVD_MODE

PWREx Wakeup Pins

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

Data Fields

- *uint32_t PLLState*
- *uint32_t PLLSource*
- *uint32_t PLLMUL*
- *uint32_t PREDIV*

Field Documentation

- ***uint32_t RCC_PLLInitTypeDef::PLLState***
The new state of the PLL. This parameter can be a value of [*RCC_PLL_Config*](#)
- ***uint32_t RCC_PLLInitTypeDef::PLLSource***
PLLSource: PLL entry clock source. This parameter must be a value of [*RCC_PLL_Clock_Source*](#)
- ***uint32_t RCC_PLLInitTypeDef::PLLMUL***
PLLMUL: Multiplication factor for PLL VCO input clock This parameter must be a value of [*RCC_PLL_Multiplication_Factor*](#)
- ***uint32_t RCC_PLLInitTypeDef::PREDIV***
PREDIV: Predivision factor for PLL VCO input clock This parameter must be a value of [*RCC_PLL_Prediv_Factor*](#)

30.1.2 RCC_OscInitTypeDef

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_OsclInitTypeDef::HSEState`**
The new state of the HSE. This parameter can be a value of [`RCC_HSE_Config`](#)
- **`uint32_t RCC_OsclInitTypeDef::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 [`RCCEx_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`

Data Fields

- **`uint32_t ClockType`**
- **`uint32_t SYSCLKSource`**
- **`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::SYSCLKSource`**
The clock source (SYSCLKS) used as system clock. This parameter can be a value of [`RCC_System_Clock_Source`](#)
- **`uint32_t RCC_ClkInitTypeDef::AHBCLKDivider`**
The AHB clock (HCLK) divider. This clock is derived from the system clock
(SYSCLK). This parameter can be a value of [`RCC_AHB_Clock_Source`](#)
- **`uint32_t RCC_ClkInitTypeDef::APB1CLKDivider`**
The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK).
This parameter can be a value of [`RCC_APB1_Clock_Source`](#)

30.2 RCC Firmware driver API description

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 enabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHB) and Low speed (APB) busses; all peripherals mapped on these busses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in input floating state, except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB busses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals whose clocks are not derived from the System clock (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 registers.

- This delay depends on the peripheral mapping.
 - AHB & APB peripherals, 1 dummy read is necessary

Workarounds:

1. For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each `__HAL_RCC_PPP_CLK_ENABLE()` macro.

30.2.3 Initialization and de-initialization function

This section provides 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 of the table below.
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)	$24 < \text{HCLK} \leq 48$

This section contains the following APIs:

- [`HAL_RCC_DeInit\(\)`](#)
- [`HAL_RCC_OscConfig\(\)`](#)
- [`HAL_RCC_ClockConfig\(\)`](#)

30.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

This section contains the following APIs:

- [`HAL_RCC_MCOConfig\(\)`](#)
- [`HAL_RCC_EnableCSS\(\)`](#)
- [`HAL_RCC_DisableCSS\(\)`](#)
- [`HAL_RCC_GetSysClockFreq\(\)`](#)

- [***HAL_RCC_GetHCLKFreq\(\)***](#)
- [***HAL_RCC_GetPCLK1Freq\(\)***](#)
- [***HAL_RCC_GetOscConfig\(\)***](#)
- [***HAL_RCC_GetClockConfig\(\)***](#)
- [***HAL_RCC_NMI_IRQHandler\(\)***](#)
- [***HAL_RCC_CSSCallback\(\)***](#)

30.2.5 HAL_RCC_DelInit

Function Name	void HAL_RCC_DelInit (void)
Function Description	Resets the RCC clock configuration to the default reset state.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE and PLL OFFAHB, APB1 prescaler set to 1.CSS and MCO1 OFFAll interrupts disabled • This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

30.2.6 HAL_RCC_OscConfig

Function Name	HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)
Function Description	Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.
Parameters	<ul style="list-style-type: none"> • RCC_OscInitStruct: pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • The PLL is not disabled when used as system clock.

30.2.7 HAL_RCC_ClockConfig

Function Name	HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)
Function Description	Initializes the CPU, AHB and APB busses clocks according to the specified parameters in the RCC_ClkInitStruct.
Parameters	<ul style="list-style-type: none"> • RCC_ClkInitStruct: pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral. • FLatency: FLASH Latency This parameter can be one of the following values: FLASH_LATENCY_0: FLASH 0 Latency cycleFLASH_LATENCY_1: FLASH 1 Latency cycle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function • The HSI is used (enabled by hardware) as system clock source after startup from Reset, wake-up from STOP and

STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).

- A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source will be ready. You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source.

30.2.8 HAL_RCC_MCOConfig

Function Name	<code>void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOsource, uint32_t RCC_MCODiv)</code>
Function Description	Selects the clock source to output on MCO pin.
Parameters	<ul style="list-style-type: none"> • RCC_MCOx: specifies the output direction for the clock source. This parameter can be one of the following values: RCC_MCO: Clock source to output on MCO1 pin(PA8). • RCC_MCOsource: specifies the clock source to output. This parameter can be one of the following values: RCC_MCOSOURCE_HSI: HSI selected as MCO clockRCC_MCOSOURCE_HSE: HSE selected as MCO clockRCC_MCOSOURCE_LSI: LSI selected as MCO clockRCC_MCOSOURCE_LSE: LSE selected as MCO clockRCC_MCOSOURCE_PLLCLK_NODIV: PLLCLK selected as MCO clock (not applicable to STM32F05x devices) RCC_MCOSOURCE_PLLCLK_DIV2: PLLCLK Divided by 2 selected as MCO clockRCC_MCOSOURCE_SYSCLK: System Clock selected as MCO clockRCC_MCOSOURCE_HSI14: HSI14 selected as MCO clockRCC_MCOSOURCE_HSI48: HSI48 selected as MCO clock • RCC_MCODiv: specifies the MCO DIV. This parameter can be one of the following values: RCC_MCODIV_1: no division applied to MCO clock
Return values	<ul style="list-style-type: none"> • None
Notes	MCO pin should be configured in alternate function mode.

30.2.9 HAL_RCC_EnableCSS

Function Name	<code>void HAL_RCC_EnableCSS (void)</code>
Function Description	Enables the Clock Security System.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M0 NMI (Non-Maskable Interrupt) exception vector.

30.2.10 HAL_RCC_DisableCSS

Function Name	void HAL_RCC_DisableCSS (void)
Function Description	Disables the Clock Security System.
Return values	<ul style="list-style-type: none"> None

30.2.11 HAL_RCC_GetSysClockFreq

Function Name	uint32_t HAL_RCC_GetSysClockFreq (void)
Function Description	Returns the SYSCLK frequency.
Return values	<ul style="list-style-type: none"> SYSCLK frequency
Notes	<ul style="list-style-type: none"> The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source: If SYSCLK source is HSI, function returns values based on HSI_VALUE(*) If SYSCLK source is HSE, function returns a value based on HSE_VALUE divided by PREDIV factor(**) 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 . (*) HSI_VALUE is a constant defined in stm32f0xx_hal_conf.h file (default value 8 MHz) but the real value may vary depending on the variations in voltage and temperature. (**) 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. 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.

30.2.12 HAL_RCC_GetHCLKFreq

Function Name	uint32_t HAL_RCC_GetHCLKFreq (void)
Function Description	Returns the HCLK frequency.
Return values	<ul style="list-style-type: none"> HCLK frequency
Notes	<ul style="list-style-type: none"> Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect. The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function

30.2.13 HAL_RCC_GetPCLK1Freq

Function Name	uint32_t HAL_RCC_GetPCLK1Freq (void)
Function Description	Returns the PCLK1 frequency.
Return values	<ul style="list-style-type: none"> • PCLK1 frequency
Notes	<ul style="list-style-type: none"> • Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

30.2.14 HAL_RCC_GetOscConfig

Function Name	void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)
Function Description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> • RCC_OscInitStruct: pointer to an RCC_OscInitTypeDef structure that will be configured.
Return values	<ul style="list-style-type: none"> • None

30.2.15 HAL_RCC_GetClockConfig

Function Name	void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)
Function Description	Get the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> • RCC_ClkInitStruct: pointer to an RCC_ClkInitTypeDef structure that contains the current clock configuration. • pFLatency: Pointer on the Flash Latency.
Return values	<ul style="list-style-type: none"> • None

30.2.16 HAL_RCC_NMI_IRQHandler

Function Name	void HAL_RCC_NMI_IRQHandler (void)
Function Description	This function handles the RCC CSS interrupt request.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This API should be called under the NMI_Handler().

30.2.17 HAL_RCC_CSSCallback

Function Name	void HAL_RCC_CSSCallback (void)
Function Description	RCC Clock Security System interrupt callback.
Return values	<ul style="list-style-type: none"> • none

30.3 RCC Firmware driver defines

30.3.1 RCC

RCC AHB Clock Enable Disable

```
_HAL_RCC_GPIOA_CLK_ENABLE  
_HAL_RCC_GPIOB_CLK_ENABLE  
_HAL_RCC_GPIOC_CLK_ENABLE  
_HAL_RCC_GPIOF_CLK_ENABLE  
_HAL_RCC_CRC_CLK_ENABLE  
_HAL_RCC_DMA1_CLK_ENABLE  
_HAL_RCC_SRAM_CLK_ENABLE  
_HAL_RCC_FLITF_CLK_ENABLE  
_HAL_RCC_GPIOA_CLK_DISABLE  
_HAL_RCC_GPIOB_CLK_DISABLE  
_HAL_RCC_GPIOC_CLK_DISABLE  
_HAL_RCC_GPIOF_CLK_DISABLE  
_HAL_RCC_CRC_CLK_DISABLE  
_HAL_RCC_DMA1_CLK_DISABLE  
_HAL_RCC_SRAM_CLK_DISABLE  
_HAL_RCC_FLITF_CLK_DISABLE
```

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
```

RCC AHB Force Release Reset

```
_HAL_RCC_AHB_FORCE_RESET  
_HAL_RCC_GPIOA_FORCE_RESET  
_HAL_RCC_GPIOB_FORCE_RESET  
_HAL_RCC_GPIOC_FORCE_RESET  
_HAL_RCC_GPIOF_FORCE_RESET  
_HAL_RCC_AHB_RELEASE_RESET
```

_HAL_RCC_GPIOA_RELEASE_RESET
_HAL_RCC_GPIOB_RELEASE_RESET
_HAL_RCC_GPIOC_RELEASE_RESET
_HAL_RCC_GPIOF_RELEASE_RESET

AHB Peripheral Clock Enable Disable Status

_HAL_RCC_GPIOA_IS_CLK_ENABLED
_HAL_RCC_GPIOB_IS_CLK_ENABLED
_HAL_RCC_GPIOC_IS_CLK_ENABLED
_HAL_RCC_GPIOF_IS_CLK_ENABLED
_HAL_RCC_CRC_IS_CLK_ENABLED
_HAL_RCC_DMA1_IS_CLK_ENABLED
_HAL_RCC_SRAM_IS_CLK_ENABLED
_HAL_RCC_FLITF_IS_CLK_ENABLED
_HAL_RCC_GPIOA_IS_CLK_DISABLED
_HAL_RCC_GPIOB_IS_CLK_DISABLED
_HAL_RCC_GPIOC_IS_CLK_DISABLED
_HAL_RCC_GPIOF_IS_CLK_DISABLED
_HAL_RCC_CRC_IS_CLK_DISABLED
_HAL_RCC_DMA1_IS_CLK_DISABLED
_HAL_RCC_SRAM_IS_CLK_DISABLED
_HAL_RCC_FLITF_IS_CLK_DISABLED

RCC APB1 Clock Enable Disable

_HAL_RCC_TIM3_CLK_ENABLE
_HAL_RCC_TIM14_CLK_ENABLE
_HAL_RCC_WWDG_CLK_ENABLE
_HAL_RCC_I2C1_CLK_ENABLE
_HAL_RCC_PWR_CLK_ENABLE
_HAL_RCC_TIM3_CLK_DISABLE
_HAL_RCC_TIM14_CLK_DISABLE
_HAL_RCC_WWDG_CLK_DISABLE
_HAL_RCC_I2C1_CLK_DISABLE
_HAL_RCC_PWR_CLK_DISABLE

RCC APB1 Clock Source

RCC_HCLK_DIV1
RCC_HCLK_DIV2
RCC_HCLK_DIV4

RCC_HCLK_DIV8

RCC_HCLK_DIV16

RCC APB1 Force Release Reset

_HAL_RCC_APB1_FORCE_RESET

_HAL_RCC_TIM3_FORCE_RESET

_HAL_RCC_TIM14_FORCE_RESET

_HAL_RCC_WWDG_FORCE_RESET

_HAL_RCC_I2C1_FORCE_RESET

_HAL_RCC_PWR_FORCE_RESET

_HAL_RCC_APB1_RELEASE_RESET

_HAL_RCC_TIM3_RELEASE_RESET

_HAL_RCC_TIM14_RELEASE_RESET

_HAL_RCC_WWDG_RELEASE_RESET

_HAL_RCC_I2C1_RELEASE_RESET

_HAL_RCC_PWR_RELEASE_RESET

APB1 Peripheral Clock Enable Disable Status

_HAL_RCC_TIM3_IS_CLK_ENABLED

_HAL_RCC_TIM14_IS_CLK_ENABLED

_HAL_RCC_WWDG_IS_CLK_ENABLED

_HAL_RCC_I2C1_IS_CLK_ENABLED

_HAL_RCC_PWR_IS_CLK_ENABLED

_HAL_RCC_TIM3_IS_CLK_DISABLED

_HAL_RCC_TIM14_IS_CLK_DISABLED

_HAL_RCC_WWDG_IS_CLK_DISABLED

_HAL_RCC_I2C1_IS_CLK_DISABLED

_HAL_RCC_PWR_IS_CLK_DISABLED

RCC APB2 Clock Enable Disable

_HAL_RCC_SYSCFG_CLK_ENABLE

_HAL_RCC_ADC1_CLK_ENABLE

_HAL_RCC_TIM1_CLK_ENABLE

_HAL_RCC_SPI1_CLK_ENABLE

_HAL_RCC_TIM16_CLK_ENABLE

_HAL_RCC_TIM17_CLK_ENABLE

_HAL_RCC_USART1_CLK_ENABLE

_HAL_RCC_DBGMCU_CLK_ENABLE

_HAL_RCC_SYSCFG_CLK_DISABLE

```
_HAL_RCC_ADC1_CLK_DISABLE  
_HAL_RCC_TIM1_CLK_DISABLE  
_HAL_RCC_SPI1_CLK_DISABLE  
_HAL_RCC_TIM16_CLK_DISABLE  
_HAL_RCC_TIM17_CLK_DISABLE  
_HAL_RCC_USART1_CLK_DISABLE  
_HAL_RCC_DBGMCU_CLK_DISABLE
```

RCC APB2 Force Release Reset

```
_HAL_RCC_APB2_FORCE_RESET  
_HAL_RCC_SYSCFG_FORCE_RESET  
_HAL_RCC_ADC1_FORCE_RESET  
_HAL_RCC_TIM1_FORCE_RESET  
_HAL_RCC_SPI1_FORCE_RESET  
_HAL_RCC_USART1_FORCE_RESET  
_HAL_RCC_TIM16_FORCE_RESET  
_HAL_RCC_TIM17_FORCE_RESET  
_HAL_RCC_DBGMCU_FORCE_RESET  
_HAL_RCC_APB2_RELEASE_RESET  
_HAL_RCC_SYSCFG_RELEASE_RESET  
_HAL_RCC_ADC1_RELEASE_RESET  
_HAL_RCC_TIM1_RELEASE_RESET  
_HAL_RCC_SPI1_RELEASE_RESET  
_HAL_RCC_USART1_RELEASE_RESET  
_HAL_RCC_TIM16_RELEASE_RESET  
_HAL_RCC_TIM17_RELEASE_RESET  
_HAL_RCC_DBGMCU_RELEASE_RESET
```

APB2 Peripheral Clock Enable Disable Status

```
_HAL_RCC_SYSCFG_IS_CLK_ENABLED  
_HAL_RCC_ADC1_IS_CLK_ENABLED  
_HAL_RCC_TIM1_IS_CLK_ENABLED  
_HAL_RCC_SPI1_IS_CLK_ENABLED  
_HAL_RCC_TIM16_IS_CLK_ENABLED  
_HAL_RCC_TIM17_IS_CLK_ENABLED  
_HAL_RCC_USART1_IS_CLK_ENABLED  
_HAL_RCC_DBGMCU_IS_CLK_ENABLED  
_HAL_RCC_SYSCFG_IS_CLK_DISABLED
```

`_HAL_RCC_ADC1_IS_CLK_DISABLED`
`_HAL_RCC_TIM1_IS_CLK_DISABLED`
`_HAL_RCC_SPI1_IS_CLK_DISABLED`
`_HAL_RCC_TIM16_IS_CLK_DISABLED`
`_HAL_RCC_TIM17_IS_CLK_DISABLED`
`_HAL_RCC_USART1_IS_CLK_DISABLED`
`_HAL_RCC_DBGMCU_IS_CLK_DISABLED`

Flags

`RCC_FLAG_HSIRDY`
`RCC_FLAG_HSERDY`
`RCC_FLAG_PLLRDY`
`RCC_FLAG_HSI14RDY`
`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_FLAG_LSERDY`
`RCC_FLAG_HSI48RDY`

Flags Interrupts Management

`_HAL_RCC_ENABLE_IT`

Description:

- Enable RCC interrupt.

Parameters:

- `_INTERRUPT_`: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
 - `RCC_IT_LSIRDY`: LSI ready interrupt
 - `RCC_IT_LSERDY`: LSE ready interrupt
 - `RCC_IT_HSIRDY`: HSI ready interrupt
 - `RCC_IT_HSERDY`: HSE ready interrupt
 - `RCC_IT_PLLRDY`: main PLL ready interrupt
 - `RCC_IT_HSI14RDY`: HSI14 ready interrupt

- interrupt enable
- RCC_IT_HSI48RDY: HSI48 ready interrupt enable (only applicable to STM32F0X2 USB devices)

_HAL_RCC_DISABLE_IT**Description:**

- Disable RCC interrupt.

Parameters:

- _INTERRUPT_: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
 - RCC_IT_LSIRDY: LSI ready interrupt
 - RCC_IT_LSERDY: LSE ready interrupt
 - RCC_IT_HSIRDY: HSI ready interrupt
 - RCC_IT_HSERDY: HSE ready interrupt
 - RCC_IT_PLLRDY: main PLL ready interrupt
 - RCC_IT_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.

Parameters:

- _INTERRUPT_: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
 - RCC_IT_LSIRDY: LSI ready interrupt.
 - RCC_IT_LSERDY: LSE ready interrupt.
 - RCC_IT_HSIRDY: HSI ready interrupt.
 - RCC_IT_HSERDY: HSE ready interrupt.
 - RCC_IT_PLLRDY: Main PLL ready interrupt.
 - RCC_IT_CSS: Clock Security System interrupt
 - RCC_IT_HSI14RDY: HSI14 ready interrupt enable
 - RCC_IT_HSI48RDY: HSI48 ready interrupt enable (only applicable to STM32F0X2 USB devices)

_HAL_RCC_GET_IT**Description:**

- Check the RCC's interrupt has occurred or not.

Parameters:

- _INTERRUPT_: specifies the RCC

interrupt source to check. This parameter can be one of the following values:

- RCC_IT_LSIRDY: LSI ready interrupt.
- RCC_IT_LSERDY: LSE ready interrupt.
- RCC_IT_HSIRDY: HSI ready interrupt.
- RCC_IT_HSERDY: HSE ready interrupt.
- RCC_IT_PLLRDY: Main PLL ready interrupt.
- RCC_IT_CSS: Clock Security System interrupt
- RCC_IT_HSI14RDY: HSI14 ready interrupt enable
- RCC_IT_HSI48RDY: HSI48 ready interrupt enable (only applicable to STM32F0X2 USB devices)

Return value:

- The new state of __INTERRUPT__ (TRUE or FALSE).

`__HAL_RCC_CLEAR_RESET_FLAGS`

The reset flags are: RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST, RCC_FLAG_LPWRRST

`__HAL_RCC_GET_FLAG`

Description:

- Check RCC flag is set or not.

Parameters:

- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - RCC_FLAG_HSIRDY: HSI oscillator clock ready.
 - RCC_FLAG_HSERDY: HSE oscillator clock ready.
 - RCC_FLAG_PLLRDY: Main PLL clock ready.
 - RCC_FLAG_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).

Get Clock source`__HAL_RCC_SYSCLK_CONFIG`**Description:**

- Macro to configure the system clock source.

Parameters:

- `RCC_SYSCLKSOURCE`: specifies the system clock source. This parameter can be one of the following values:
 - `RCC_SYSCLKSOURCE_MSI`: MSI oscillator is used as system clock source.
 - `RCC_SYSCLKSOURCE_HSI`: HSI oscillator is used as system clock source.
 - `RCC_SYSCLKSOURCE_HSE`: HSE oscillator is used as system clock source.
 - `RCC_SYSCLKSOURCE_PLLCLK`: PLL output is used as system clock source.

`__HAL_RCC_GET_SYSCLK_SOURCE`**Description:**

- Macro to get the clock source used as system clock.

Return value:

- The clock source used as system clock. The returned value can be one of the following:
 - `RCC_SYSCLKSOURCE_STATUS_HSI`: HSI used as system clock
 - `RCC_SYSCLKSOURCE_STATUS_HSE`: HSE used as system clock
 - `RCC_SYSCLKSOURCE_STATUS_PLLCLK`: PLL used as system clock

HSE Config`RCC_HSE_OFF` HSE clock deactivation`RCC_HSE_ON` HSE clock activation`RCC_HSE_BYPASS` External clock source for HSE clock**HSE Configuration**`__HAL_RCC_HSE_CONFIG`**Description:**

- Macro to configure the External High Speed oscillator (HSE).

Parameters:

- STATE: specifies the new state of the HSE. This parameter can be one of the following values:
 - RCC_HSE_OFF: turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
 - RCC_HSE_ON: turn ON the HSE oscillator
 - RCC_HSE_BYPASS: HSE oscillator bypassed with external clock

Notes:

- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass. After enabling the HSE (RCC_HSE_ON or RCC_HSE_Bypass), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY modes. This function reset the CSSON bit, so if the Clock security system(CSS) was previously enabled you have to enable it again after calling this function.

[__HAL_RCC_HSE_PREDIV_CONFIG](#)**Description:**

- Macro to configure the External High Speed oscillator (HSE) Predivision factor for PLL.

Parameters:

- HSE_PREDIV_VALUE: specifies the division value applied to HSE. This parameter must be a number between RCC_HSE_PREDIV_DIV1 and RCC_HSE_PREDIV_DIV16.

Notes:

- Predivision factor can not be changed if PLL is used as system clock In this case, you have to select another source of the system clock, disable the PLL and then change the HSE predivision factor.

RCC HSI14 Config[RCC_HSI14_OFF](#)

RCC_HSI14_ON
RCC_HSI14_ADC_CONTROL
RCC_HSI14CALIBRATION_DEFAULT
RCC_HSI14_Configuration
__HAL_RCC_HSI14_ENABLE

Notes:

- The HSI14 is stopped by hardware when entering STOP and STANDBY modes. HSI14 can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI14. After enabling the HSI14 with __HAL_RCC_HSI14_ENABLE(), the application software should wait on HSI14RDY flag to be set indicating that HSI clock is stable and can be used as system clock source. This is not necessary if HAL_RCC_OscConfig() is used. When the HSI14 is stopped, HSI14RDY flag goes low after 6 HSI14 oscillator clock cycles.

__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.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI14 RC.

__HAL_RCC_HSI14_CALIBRATIONVALUE_ADJ
UST

HSI Config

RCC_HSI_OFF	HSI clock deactivation
RCC_HSI_ON	HSI clock activation
RCC_HSICALIBRATION_DEFAULT	

HSI Configuration`__HAL_RCC_HSI_ENABLE`**Notes:**

- The HSI is stopped by hardware when entering STOP and STANDBY modes. HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

`__HAL_RCC_HSI_DISABLE``__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST`**Description:**

- macro to adjust the Internal High Speed oscillator (HSI) calibration value.

Parameters:

- `_HSICALIBRATIONVALUE_`: specifies the calibration trimming value. (default is `RCC_HSICALIBRATION_DEFAULT`). This parameter must be a number between 0 and 0x1F.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

RCC I2C1 Clock Source

RCC_I2C1CLKSOURCE_HSI	
RCC_I2C1CLKSOURCE_SYSCLK	

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

Interrupts

RCC_IT_LSIRDY	LSI Ready Interrupt flag
RCC_IT_LSERDY	LSE Ready Interrupt flag
RCC_IT_HSIRDY	HSI Ready Interrupt flag
RCC_IT_HSERDY	HSE Ready Interrupt flag
RCC_IT_PLLRDY	PLL Ready Interrupt flag
RCC_IT_HSI14	HSI14 Ready Interrupt flag
RCC_IT_CSS	Clock Security System Interrupt flag
RCC_IT_HSI48	HSI48 Ready Interrupt flag

LSE Config

RCC_LSE_OFF	LSE clock deactivation
RCC_LSE_ON	LSE clock activation
RCC_LSE_BYPASS	External clock source for LSE clock

LSE Configuration

_HAL_RCC_LSE_CONFIG **Description:**

- Macro to configure the External Low Speed oscillator (LSE).

Parameters:

- STATE: specifies the new state of the LSE. This parameter can be one of the following values:
 - RCC_LSE_OFF: turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
 - RCC_LSE_ON: turn ON the LSE oscillator.

- RCC_LSE_BYPASS: LSE oscillator bypassed with external clock.

Notes:

- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC_LSE_ON or RCC_LSE_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

LSI Config

RCC_LSI_OFF	LSI clock deactivation
RCC_LSI_ON	LSI clock activation

LSI Configuration

__HAL_RCC_LSI_ENABLE	Notes:
----------------------	---------------

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

__HAL_RCC_LSI_DISABLE

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_MCOSOURCE_HSI48
RCC_MCOSOURCE_PLLCLK_NODIV

MCO Index

RCC_MCO1	
RCC_MCO	MCO1 to be compliant with other families with 2 MCOs

Oscillator Type

RCC_OSCILLATORTYPE_NONE

RCC_OSCILLATORTYPE_HSE
RCC_OSCILLATORTYPE_HSI
RCC_OSCILLATORTYPE_LSE
RCC_OSCILLATORTYPE_LSI
RCC_OSCILLATORTYPE_HSI14
RCC_OSCILLATORTYPE_HSI48

PLL Clock Source

RCC_PLLSOURCE_HSE HSE clock selected as PLL entry clock source
RCC_PLLSOURCE_HSI
RCC_PLLSOURCE_HSI48

PLL Config

RCC_PLL_NONE PLL is not configured
RCC_PLL_OFF PLL deactivation
RCC_PLL_ON PLL activation

PLL Configuration

__HAL_RCC_PLL_ENABLE

Notes:

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL_DISABLE

Notes:

- The main PLL can not be disabled if it is used as system clock source

__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
- __PLLMUL__: specifies the multiplication factor for PLL VCO output clock. This parameter can be one of the following values: This parameter must be a number between RCC_PLL_MUL2 and RCC_PLL_MUL16.

- `__PREDIV__`: specifies the predivider factor for PLL VCO input clock. This parameter must be a number between `RCC_PREDIV_DIV1` and `RCC_PREDIV_DIV16`.

Notes:

- This function must be used only when the main PLL is disabled.

`__HAL_RCC_GET_PLL_OSCSOURCE`**Description:**

- Get oscillator clock selected as PLL input clock.

Return value:

- The: clock source used for PLL entry. The returned value can be one of the following:
 - `RCC_PLLSOURCE_HSE`: HSE oscillator clock selected as PLL input clock

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`***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

Register offsets

RCC_OFFSET
RCC_CR_OFFSET
RCC_CFGR_OFFSET
RCC_CIR_OFFSET
RCC_BDCR_OFFSET
RCC_CSR_OFFSET

RCC RTC Clock Configuration

`_HAL_RCC_RTC_CONFIG`

Description:

- Macro to configures the RTC clock (RTCCLK).

Parameters:

- `_RTC_CLKSOURCE_`: specifies the RTC clock source. This parameter can be one of the following values:
 - `RCC_RTCCLKSOURCE_NO_CLK`: No clock selected as RTC clock
 - `RCC_RTCCLKSOURCE_LSE`: LSE selected as RTC clock
 - `RCC_RTCCLKSOURCE_LSI`: LSI selected as RTC clock
 - `RCC_RTCCLKSOURCE_HSE_DIV32`: HSE clock divided by 32

Notes:

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it can't be changed unless the Backup domain is reset using

`_HAL_RCC_BACKUPRESET_FORCE()` macro, or by a Power On Reset (POR).

- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the LSI clock and HSE clock divided by 32 is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The system must always be configured so as to get a PCLK frequency greater than or equal to the RTCCLK frequency for a proper operation of the RTC.

`_HAL_RCC_GET_RTC_SOURCE`

Description:

- macros to get the RTC clock source.

Return value:

- The: clock source can be one of the following values:
 - `RCC_RTCCLKSOURCE_NO_CLK`: No clock selected as RTC clock
 - `RCC_RTCCLKSOURCE_LSE`: LSE selected as RTC clock
 - `RCC_RTCCLKSOURCE_LSI`: LSI selected as RTC clock
 - `RCC_RTCCLKSOURCE_HSE_DIV32`: HSE clock divided by 32

`_HAL_RCC_RTC_ENABLE`

Notes:

- These macros must be used only after the RTC clock source was selected.

`_HAL_RCC_RTC_DISABLE`

Notes:

- These macros must be used only after the RTC clock source was selected.

`_HAL_RCC_BACKUPRESET_FORCE`

Notes:

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in `RCC_BDCR` register.

`_HAL_RCC_BACKUPRESET_RELEASE`

RTC Clock Source

<code>RCC_RTCCLKSOURCE_NO_CLK</code>	No clock
<code>RCC_RTCCLKSOURCE_LSE</code>	LSE oscillator clock used as RTC clock
<code>RCC_RTCCLKSOURCE_LSI</code>	LSI oscillator clock used as RTC clock
<code>RCC_RTCCLKSOURCE_HSE_DIV32</code>	HSE oscillator clock divided by 32 used as RTC clock

System Clock Source

RCC_SYSCLKSOURCE_HSI	HSI selected as system clock
RCC_SYSCLKSOURCE_HSE	HSE selected as system clock
RCC_SYSCLKSOURCE_PLLCLK	PLL selected as system clock
RCC_SYSCLKSOURCE_HSI48	

System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_HSI	HSI used as system clock
RCC_SYSCLKSOURCE_STATUS_HSE	HSE used as system clock
RCC_SYSCLKSOURCE_STATUS_PLLCLK	PLL used as system clock
RCC_SYSCLKSOURCE_STATUS_HSI48	

System Clock Type

RCC_CLOCKTYPE_SYSCLK	SYSCLK to configure
RCC_CLOCKTYPE_HCLK	HCLK to configure
RCC_CLOCKTYPE_PCLK1	PCLK1 to configure

RCC Timeout

RCC_DBP_TIMEOUT_VALUE	
RCC_LSE_TIMEOUT_VALUE	
CLOCKSWITCH_TIMEOUT_VALUE	
HSE_TIMEOUT_VALUE	
HSI_TIMEOUT_VALUE	
LSI_TIMEOUT_VALUE	
PLL_TIMEOUT_VALUE	
HSI14_TIMEOUT_VALUE	
HSI48_TIMEOUT_VALUE	

RCC USART1 Clock Source

RCC_USART1CLKSOURCE_PCLK1	
RCC_USART1CLKSOURCE_SYSCLK	
RCC_USART1CLKSOURCE_LSE	
RCC_USART1CLKSOURCE_HSI	

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`

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

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

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

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

31.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.



Important note: Care must be taken when HAL_RCCEEx_PерiphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register are set to their reset values.

This section contains the following APIs:

- [***HAL_RCCEEx_PерiphCLKConfig\(\)***](#)
- [***HAL_RCCEEx_GetPeriphCLKConfig\(\)***](#)
- [***HAL_RCCEEx_GetPeriphCLKFreq\(\)***](#)
- [***HAL_RCCEEx_CRSConfig\(\)***](#)
- [***HAL_RCCEEx_CRSSoftwareSynchronizationGenerate\(\)***](#)
- [***HAL_RCCEEx_CRSGetSynchronizationInfo\(\)***](#)
- [***HAL_RCCEEx_CRSWaitSynchronization\(\)***](#)

31.2.2 HAL_RCCEEx_PерiphCLKConfig

Function Name	<code>HAL_StatusTypeDef HAL_RCCEEx_PерiphCLKConfig (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"> • PeriphClkInit: pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks (USART, RTC, I2C, CEC and USB).
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • Care must be taken when HAL_RCCEEx_PерiphCLKConfig() 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.

31.2.3 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"> PeriphClkInit: pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks (USART, RTC, I2C, CEC and USB).
Return values	<ul style="list-style-type: none"> None

31.2.4 HAL_RCCEx_GetPeriphCLKFreq

Function Name	uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint32_t PeriphClk)
Function Description	Returns the peripheral clock frequency.
Parameters	<ul style="list-style-type: none"> PeriphClk: Peripheral clock identifier This parameter can be one of the following values: RCC_PERIPHCLK_RTC RTC peripheral clockRCC_PERIPHCLK_USART1 USART1 peripheral clockRCC_PERIPHCLK_USART2 USART2 peripheral clock (*)RCC_PERIPHCLK_USART3 USART3 peripheral clock (*)RCC_PERIPHCLK_I2C1 I2C1 peripheral clockRCC_PERIPHCLK_USB USB peripheral clock (*)RCC_PERIPHCLK_CEC CEC peripheral clock (*)
Return values	<ul style="list-style-type: none"> Frequency in Hz (0: means that no available frequency for the peripheral)
Notes	<ul style="list-style-type: none"> Returns 0 if peripheral clock is unknown (*) means that this peripheral is not present on all the STM32F0xx devices

31.2.5 HAL_RCCEx_CRSConfig

Function Name	void HAL_RCCEx_CRSConfig (RCC_CRSInitTypeDef * pInit)
Function Description	Start automatic synchronization using polling mode.
Parameters	<ul style="list-style-type: none"> pInit: Pointer on RCC_CRSInitTypeDef structure
Return values	<ul style="list-style-type: none"> None

31.2.6 HAL_RCCEx_CRSSoftwareSynchronizationGenerate

Function Name	void HAL_RCCEx_CRSSoftwareSynchronizationGenerate (void)
Function Description	Generate the software synchronization event.
Return values	<ul style="list-style-type: none"> None

31.2.7 HAL_RCCEx_CRSGetSynchronizationInfo

Function Name	void HAL_RCCEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)
Function Description	Function to return synchronization info.
Parameters	<ul style="list-style-type: none"> pSynchroInfo: Pointer on RCC_CRSSynchroInfoTypeDef structure

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

31.2.8 HAL_RCCEEx_CRSWaitSynchronization

Function Name	<code>uint32_t HAL_RCCEEx_CRSWaitSynchronization (uint32_t Timeout)</code>
Function Description	This function handles CRS Synchronization Timeout.
Parameters	<ul style="list-style-type: none"> Timeout: Duration of the timeout
Return values	<ul style="list-style-type: none"> Combination of Synchronization status This parameter can be a combination of the following values: <code>RCC_CRS_TIMEOUT</code> <code>RCC_CRS_SYNCOK</code> <code>RCC_CRS_SYNCWARM</code> <code>RCC_CRS_SYNCERR</code> <code>RCC_CRS_SYNCMISS</code> <code>RCC_CRS_TRIMOV</code>
Notes	<ul style="list-style-type: none"> Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency. If Timeout set to <code>HAL_MAX_DELAY</code>, <code>HAL_TIMEOUT</code> will be never returned.

31.3 RCCEEx Firmware driver defines

31.3.1 RCCEEx

RCCEEx CEC Clock Source

`RCC_CECCLKSOURCE_HSI`

`RCC_CECCLKSOURCE_LSE`

RCCEEx CRS ErrorLimitDefault

`RCC_CRS_ERRORLIMIT_DEFAULT` Default Frequency error limit

RCCEEx CRS Extended Features

<code>_HAL_RCC_CRS_ENABLE_FREQ_ERROR_COUNTER</code>	Description:
	<ul style="list-style-type: none"> Enables the oscillator clock for frequency error counter.

Return value:

- None

Notes:

- when the CEN bit is set the CRS_CFG register becomes write-protected.

<code>_HAL_RCC_CRS_DISABLE_FREQ_ERROR_COUNTER</code>	Description:
	<ul style="list-style-type: none"> Disables the oscillator clock for frequency error counter.

Return value:

`_HAL_RCC_CRS_ENABLE_AUTOMATIC_CALIB`

- None

Description:

- Enables the automatic hardware adjustement of TRIM bits.

Return value:

- None

Notes:

- When the AUTOTRIMEN bit is set the CRS_CFGR register becomes write-protected.

`_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

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected

synchronization on the zero value. The formula is the following:

$$\text{RELOAD} = (\text{fTARGET} / \text{fSYNC}) - 1$$

RCCEEx 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

RCCEEx CRS FreqErrorDirection

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

RCCEEx CRS HSI48CalibrationDefault

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
----------------------------------	--

RCCEEx CRS Interrupt Sources

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

RCCEEx CRS ReloadValueDefault

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).
-----------------------------	--

RCCEEx CRS Status

RCC_CRS_NONE	
RCC_CRS_TIMEOUT	

RCC_CRS_SYNCOK
 RCC_CRS_SYNCWARM
 RCC_CRS_SYNCERR
 RCC_CRS_SYNCMISS
 RCC_CRS_TRIMOV

RCCEEx CRS SynchroDivider

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

RCCEEx CRS SynchroPolarity

RCC_CRS_SYNC_POLARITY_RISING	Synchro Active on rising edge (default)
RCC_CRS_SYNC_POLARITY_FALLING	Synchro Active on falling edge

RCCEEx CRS SynchroSource

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)

RCCEEx Force Release Peripheral Reset

- __HAL_RCC_GPIOD_FORCE_RESET
- __HAL_RCC_GPIOD_RELEASE_RESET
- __HAL_RCC_GPIOE_FORCE_RESET
- __HAL_RCC_GPIOE_RELEASE_RESET
- __HAL_RCC_TSC_FORCE_RESET
- __HAL_RCC_TSC_RELEASE_RESET
- __HAL_RCC_USART2_FORCE_RESET
- __HAL_RCC_SPI2_FORCE_RESET
- __HAL_RCC_USART2_RELEASE_RESET
- __HAL_RCC_SPI2_RELEASE_RESET
- __HAL_RCC_TIM2_FORCE_RESET
- __HAL_RCC_TIM2_RELEASE_RESET
- __HAL_RCC_TIM6_FORCE_RESET
- __HAL_RCC_I2C2_FORCE_RESET

```
_HAL_RCC_TIM6_RELEASE_RESET  
_HAL_RCC_I2C2_RELEASE_RESET  
_HAL_RCC_DAC1_FORCE_RESET  
_HAL_RCC_DAC1_RELEASE_RESET  
_HAL_RCC_CEC_FORCE_RESET  
_HAL_RCC_CEC_RELEASE_RESET  
_HAL_RCC_TIM7_FORCE_RESET  
_HAL_RCC_USART3_FORCE_RESET  
_HAL_RCC_USART4_FORCE_RESET  
_HAL_RCC_TIM7_RELEASE_RESET  
_HAL_RCC_USART3_RELEASE_RESET  
_HAL_RCC_USART4_RELEASE_RESET  
_HAL_RCC_CAN1_FORCE_RESET  
_HAL_RCC_CAN1_RELEASE_RESET  
_HAL_RCC_CRS_FORCE_RESET  
_HAL_RCC_CRS_RELEASE_RESET  
_HAL_RCC_USART5_FORCE_RESET  
_HAL_RCC_USART5_RELEASE_RESET  
_HAL_RCC_TIM15_FORCE_RESET  
_HAL_RCC_TIM15_RELEASE_RESET  
_HAL_RCC_USART6_FORCE_RESET  
_HAL_RCC_USART6_RELEASE_RESET  
_HAL_RCC_USART7_FORCE_RESET  
_HAL_RCC_USART8_FORCE_RESET  
_HAL_RCC_USART7_RELEASE_RESET  
_HAL_RCC_USART8_RELEASE_RESET
```

RCCEx HSI48 Config

RCC_HSI48_OFF

RCC_HSI48_ON

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

RCCEEx 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`

RCCEx 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`

RCCEx_Peripheral_Clock_Enable_Disable

`_HAL_RCC_GPIOD_CLK_ENABLE`

`_HAL_RCC_GPIOD_CLK_DISABLE`

`_HAL_RCC_GPIOE_CLK_ENABLE`

`_HAL_RCC_GPIOE_CLK_DISABLE`

`_HAL_RCC_TSC_CLK_ENABLE`

`_HAL_RCC_TSC_CLK_DISABLE`

`_HAL_RCC_DMA2_CLK_ENABLE`

`_HAL_RCC_DMA2_CLK_DISABLE`

`_HAL_RCC_USART2_CLK_ENABLE`

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

`_HAL_RCC_USART2_CLK_DISABLE`

`_HAL_RCC_SPI2_CLK_ENABLE`

`_HAL_RCC_SPI2_CLK_DISABLE`

`_HAL_RCC_TIM2_CLK_ENABLE`

`_HAL_RCC_TIM2_CLK_DISABLE`

`_HAL_RCC_TIM6_CLK_ENABLE`

`_HAL_RCC_I2C2_CLK_ENABLE`

`_HAL_RCC_TIM6_CLK_DISABLE`

`_HAL_RCC_I2C2_CLK_DISABLE`

`_HAL_RCC_DAC1_CLK_ENABLE`

`_HAL_RCC_DAC1_CLK_DISABLE`

`_HAL_RCC_CEC_CLK_ENABLE`

```
_HAL_RCC_CEC_CLK_DISABLE  
_HAL_RCC_TIM7_CLK_ENABLE  
_HAL_RCC_USART3_CLK_ENABLE  
_HAL_RCC_USART4_CLK_ENABLE  
_HAL_RCC_TIM7_CLK_DISABLE  
_HAL_RCC_USART3_CLK_DISABLE  
_HAL_RCC_USART4_CLK_DISABLE  
_HAL_RCC_CAN1_CLK_ENABLE  
_HAL_RCC_CAN1_CLK_DISABLE  
_HAL_RCC_CRS_CLK_ENABLE  
_HAL_RCC_CRS_CLK_DISABLE  
_HAL_RCC_USART5_CLK_ENABLE  
_HAL_RCC_USART5_CLK_DISABLE  
_HAL_RCC_TIM15_CLK_ENABLE
```

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

```
_HAL_RCC_TIM15_CLK_DISABLE  
_HAL_RCC_USART6_CLK_ENABLE  
_HAL_RCC_USART6_CLK_DISABLE  
_HAL_RCC_USART7_CLK_ENABLE  
_HAL_RCC_USART8_CLK_ENABLE  
_HAL_RCC_USART7_CLK_DISABLE  
_HAL_RCC_USART8_CLK_DISABLE
```

Peripheral Clock Enable Disable Status

```
_HAL_RCC_GPIOD_IS_CLK_ENABLED  
_HAL_RCC_GPIOD_IS_CLK_DISABLED  
_HAL_RCC_GPIOE_IS_CLK_ENABLED  
_HAL_RCC_GPIOE_IS_CLK_DISABLED  
_HAL_RCC_TSC_IS_CLK_ENABLED  
_HAL_RCC_TSC_IS_CLK_DISABLED  
_HAL_RCC_DMA2_IS_CLK_ENABLED  
_HAL_RCC_DMA2_IS_CLK_DISABLED  
_HAL_RCC_USART2_IS_CLK_ENABLED  
_HAL_RCC_USART2_IS_CLK_DISABLED  
_HAL_RCC_SPI2_IS_CLK_ENABLED
```

```
_HAL_RCC_SPI2_IS_CLK_DISABLED  
_HAL_RCC_TIM2_IS_CLK_ENABLED  
_HAL_RCC_TIM2_IS_CLK_DISABLED  
_HAL_RCC_TIM6_IS_CLK_ENABLED  
_HAL_RCC_I2C2_IS_CLK_ENABLED  
_HAL_RCC_TIM6_IS_CLK_DISABLED  
_HAL_RCC_I2C2_IS_CLK_DISABLED  
_HAL_RCC_DAC1_IS_CLK_ENABLED  
_HAL_RCC_DAC1_IS_CLK_DISABLED  
_HAL_RCC_CEC_IS_CLK_ENABLED  
_HAL_RCC_CEC_IS_CLK_DISABLED  
_HAL_RCC_TIM7_IS_CLK_ENABLED  
_HAL_RCC_USART3_IS_CLK_ENABLED  
_HAL_RCC_USART4_IS_CLK_ENABLED  
_HAL_RCC_TIM7_IS_CLK_DISABLED  
_HAL_RCC_USART3_IS_CLK_DISABLED  
_HAL_RCC_USART4_IS_CLK_DISABLED  
_HAL_RCC_CAN1_IS_CLK_ENABLED  
_HAL_RCC_CAN1_IS_CLK_DISABLED  
_HAL_RCC_CRS_IS_CLK_ENABLED  
_HAL_RCC_CRS_IS_CLK_DISABLED  
_HAL_RCC_USART5_IS_CLK_ENABLED  
_HAL_RCC_USART5_IS_CLK_DISABLED  
_HAL_RCC_TIM15_IS_CLK_ENABLED  
_HAL_RCC_TIM15_IS_CLK_DISABLED  
_HAL_RCC_USART6_IS_CLK_ENABLED  
_HAL_RCC_USART6_IS_CLK_DISABLED  
_HAL_RCC_USART7_IS_CLK_ENABLED  
_HAL_RCC_USART8_IS_CLK_ENABLED  
_HAL_RCC_USART7_IS_CLK_DISABLED  
_HAL_RCC_USART8_IS_CLK_DISABLED
```

RCCEx 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:

- `_MCOCLKSource_`: 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`

RCCEx USART2 Clock Source

`RCC_USART2CLKSOURCE_PCLK1`

`RCC_USART2CLKSOURCE_SYSCLK`

`RCC_USART2CLKSOURCE_LSE`

`RCC_USART2CLKSOURCE_HSI`

RCCEx USART3 Clock Source

`RCC_USART3CLKSOURCE_PCLK1`

`RCC_USART3CLKSOURCE_SYSCLK`

`RCC_USART3CLKSOURCE_LSE`

`RCC_USART3CLKSOURCE_HSI`

32 HAL RTC Generic Driver

32.1 RTC Firmware driver registers structures

32.1.1 RTC_InitTypeDef

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 = 0xFFFF
- ***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

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

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

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

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

32.2.1 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.

32.2.2 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 wake-up, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wake-up mode), by using the RTC alarm or the RTC wake-up events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wake-up from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

32.2.3 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
 - A 7-bit asynchronous prescaler and a 15-bit synchronous prescaler.
 - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.

2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wake-up from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers. The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- [*HAL_RTC_Init\(\)*](#)
- [*HAL_RTC_DeInit\(\)*](#)
- [*HAL_RTC_MspInit\(\)*](#)
- [*HAL_RTC_MspDeInit\(\)*](#)

32.2.4 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

- [*HAL_RTC_SetTime\(\)*](#)
- [*HAL_RTC_GetTime\(\)*](#)
- [*HAL_RTC_SetDate\(\)*](#)
- [*HAL_RTC_GetDate\(\)*](#)

32.2.5 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:

- [*HAL_RTC_SetAlarm\(\)*](#)
- [*HAL_RTC_SetAlarm_IT\(\)*](#)
- [*HAL_RTC_DeactivateAlarm\(\)*](#)
- [*HAL_RTC_GetAlarm\(\)*](#)
- [*HAL_RTC_AlarmIRQHandler\(\)*](#)
- [*HAL_RTC_AlarmAEventCallback\(\)*](#)
- [*HAL_RTC_PollForAlarmAEvent\(\)*](#)

32.2.6 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

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

32.2.7 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

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

32.2.8 HAL_RTC_Init

Function Name	HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)
Function Description	Initialize the RTC according to the specified parameters in the RTC_InitTypeDef structure and initialize the associated handle.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• HAL status

32.2.9 HAL_RTC_DeInit

Function Name	HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)
Function Description	Deinitialize the RTC peripheral.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• HAL status

Notes

- This function doesn't reset the RTC Backup Data registers.

32.2.10 HAL_RTC_MspInit

Function Name	void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)
Function Description	Initialize the RTC MSP.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None

32.2.11 HAL_RTC_MspDeInit

Function Name	void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)
Function Description	Deinitialize the RTC MSP.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None

32.2.12 HAL_RTC_SetTime

Function Name	HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
Function Description	Set RTC current time.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle• sTime: Pointer to Time structure• Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format

- Return values
- HAL status

32.2.13 HAL_RTC_GetTime

Function Name	<code>HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)</code>
Function Description	Get RTC current time.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • sTime: Pointer to Time structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read to ensure consistency between the time and date values.

32.2.14 HAL_RTC_SetDate

Function Name	<code>HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)</code>
Function Description	Set RTC current date.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • sDate: Pointer to date structure • Format: specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

32.2.15 HAL_RTC_GetDate

Function Name	<code>HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)</code>
Function Description	Get RTC current date.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • sDate: Pointer to Date structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN : Binary data formatRTC_FORMAT_BCD : BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the

time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

32.2.16 HAL_RTC_SetAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
Function Description	Set the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • sAlarm: Pointer to Alarm structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

32.2.17 HAL_RTC_SetAlarm_IT

Function Name	HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
Function Description	Set the specified RTC Alarm with Interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • sAlarm: Pointer to Alarm structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()). • The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

32.2.18 HAL_RTC_DeactivateAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)
Function Description	Deactivate the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • Alarm: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmA
Return values	<ul style="list-style-type: none"> • HAL status

32.2.19 HAL_RTC_GetAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_GetAlarm
---------------	---

**(RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,
uint32_t Alarm, uint32_t Format)**

Function Description	Get the RTC Alarm value and masks.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • sAlarm: Pointer to Date structure • Alarm: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmA • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

32.2.20 HAL_RTC_AlarmIRQHandler

Function Name	void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	Handle Alarm interrupt request.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • None

32.2.21 HAL_RTC_AlarmAEventCallback

Function Name	void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Alarm A callback.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • None

32.2.22 HAL_RTC_PollForAlarmAEvent

Function Name	HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	Handle AlarmA Polling request.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

32.2.23 HAL_RTC_WaitForSynchro

Function Name	HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)
Function Description	Wait until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • HAL status

Notes	<ul style="list-style-type: none"> • The RTC Resynchronization mode is write protected, use the <code>_HAL_RTC_WRITEPROTECTION_DISABLE()</code> before calling this function. • To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.
-------	--

32.2.24 HAL_RTC_GetState

Function Name	<code>HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)</code>
Function Description	Return the RTC handle state.
Parameters	<ul style="list-style-type: none"> • <code>hrtc</code>: RTC handle
Return values	<ul style="list-style-type: none"> • HAL state

32.3 RTC Firmware driver defines

32.3.1 RTC

RTC Alarm Date WeekDay Definitions

`RTC_ALARMDATEWEEKDAYSEL_DATE`
`RTC_ALARMDATEWEEKDAYSEL_WEEKDAY`

RTC Alarm Mask Definitions

`RTC_ALARMMASK_NONE`
`RTC_ALARMMASK_DATEWEEKDAY`
`RTC_ALARMMASK_HOURS`
`RTC_ALARMMASK_MINUTES`
`RTC_ALARMMASK_SECONDS`
`RTC_ALARMMASK_ALL`

RTC Alarms Definitions

`RTC_ALARM_A`

RTC Alarm Sub Seconds Masks Definitions

<code>RTC_ALARMSUBSECONDMASK_ALL</code>	All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm
<code>RTC_ALARMSUBSECONDMASK_SS14_1</code>	SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.
<code>RTC_ALARMSUBSECONDMASK_SS14_2</code>	SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_3</code>	SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_4	SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_5	SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_6	SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_7	SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_8	SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_9	SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_10	SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_11	SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_12	SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_13	SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared
RTC_ALARMSUBSECONDMASK_SS14	SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared
RTC_ALARMSUBSECONDMASK_NONE	SS[14:0] are compared and must match to activate alarm.

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC DayLight Saving Definitions

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

RTC Exported Macros_HAL_RTC_RESET_HANDLE_STATE**Description:**

- Reset RTC handle state.

Parameters:

- _HANDLE_: RTC handle.

Return value:

- None

_HAL_RTC_WRITEPROTECTION_DISABLE**Description:**

- Disable the write

protection for RTC registers.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_WRITEPROTECTION_ENABLE`

Description:

- Enable the write protection for RTC registers.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_ALARMA_ENABLE`

Description:

- Enable the RTC ALARMA peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_ALARMA_DISABLE`

Description:

- Disable the RTC ALARMA peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_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.
- `__INTERRUPT__`: specifies the RTC Alarm interrupt to check. This parameter can be:
 - RTC_IT_ALRA:
Alarm A interrupt

Return value:

- None

`__HAL_RTC_ALARM_GET_IT_SOURCE`

Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- HANDLE: specifies the RTC handle.
- INTERRUPT: specifies the RTC Alarm interrupt sources to check. This parameter can be:
 - RTC_IT_ALRA: Alarm A interrupt

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 check. This parameter can be:
 - RTC_FLAG_ALRAF
 - RTC_FLAG_ALRAW
 - F

Return value:

- None

_HAL_RTC_ALARM_CLEAR_FLAG**Description:**

- Clear the RTC Alarm's pending flags.

Parameters:

- HANDLE: specifies the RTC handle.
- FLAG: specifies the RTC Alarm Flag sources to clear. This parameter can be:
 - RTC_FLAG_ALRAF

Return value:

- None

_HAL_RTC_ALARM_EXTI_ENABLE_IT**Description:**

- Enable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

Description:

- Disable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

Description:

- Enable event on the RTC Alarm associated Exti line.

Return value:

- None.

Description:

- Disable event on the RTC Alarm associated Exti line.

Return value:

- None.

Description:

- Enable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

Description:

- Disable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

Description:

- Enable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

Description:

- Disable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

Description:

- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

Description:

- Disable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

Description:

- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

Description:

- Clear the RTC Alarm associated Exti line flag.

Return value:

- None.

Description:

- Generate a Software interrupt on RTC Alarm associated Exti line.

Return value:

- None.

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
RTC Input parameter format definitions
RTC_FORMAT_BIN
RTC_FORMAT_BCD
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 Private macros to check input parameters
IS_RTC_HOUR_FORMAT
IS_RTC_OUTPUT_POL
IS_RTC_OUTPUT_TYPE
IS_RTC_HOUR12
IS_RTC_HOUR24
IS_RTC_ASYNCH_PREDIV
IS_RTC_SYNCH_PREDIV
IS_RTC_MINUTES
IS_RTC_SECONDS
IS_RTC_HOURFORMAT12
IS_RTC_DAYLIGHT_SAVING
IS_RTC_STORE_OPERATION
IS_RTC_FORMAT
IS_RTC_YEAR
IS_RTC_MONTH

IS_RTC_DATE
IS_RTC_WEEKDAY
IS_RTC_ALARM_DATE_WEEKDAY_DATE
IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY
IS_RTC_ALARM_DATE_WEEKDAY_SEL
IS_RTC_ALARM_MASK
IS_RTC_ALARM
IS_RTC_ALARM_SUB_SECOND_VALUE
IS_RTC_ALARM_SUB_SECOND_MASK

RTC Month Date Definitions

RTC_MONTH_JANUARY
RTC_MONTH_FEBRUARY
RTC_MONTH_MARCH
RTC_MONTH_APRIIL
RTC_MONTH_MAY
RTC_MONTH_JUNE
RTC_MONTH_JULY
RTC_MONTH_AUGUST
RTC_MONTH_SEPTEMBER
RTC_MONTH_OCTOBER
RTC_MONTH_NOVEMBER
RTC_MONTH_DECEMBER

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH
RTC_OUTPUT_POLARITY_LOW

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_OPENDRAIN
RTC_OUTPUT_TYPE_PUSH_PULL

RTC Store Operation Definitions

RTC_STOREOPERATION_RESET
RTC_STOREOPERATION_SET

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY
RTC_WEEKDAY_TUESDAY
RTC_WEEKDAY_WEDNESDAY
RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY
RTC_WEEKDAY_SATURDAY
RTC_WEEKDAY_SUNDAY

33 HAL RTC Extension Driver

33.1 RTCEEx Firmware driver registers structures

33.1.1 RTC_TamperTypeDef

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

33.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wake-up configuration



Not available on F030x4/x6/x8 and F070x6

TimeStamp configuration

- Configure the RTC_AF trigger and enable the RTC TimeStamp using the HAL_RTCEx_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTCEx_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, precharge or discharge and Pull-UP using the HAL_RTCEx_SetTamper() function. You can configure RTC Tamper in interrupt mode using HAL_RTCEx_SetTamper_IT() function.

Backup Data Registers configuration



Not available on F030x6/x8/xC and F070x6/xB (F0xx Value Line devices)

33.2.2

RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

This section contains the following APIs:

- [*HAL_RTCEx_SetTimeStamp\(\)*](#)
- [*HAL_RTCEx_SetTimeStamp_IT\(\)*](#)
- [*HAL_RTCEx_DeactivateTimeStamp\(\)*](#)
- [*HAL_RTCEx_GetTimeStamp\(\)*](#)
- [*HAL_RTCEx_SetTamper\(\)*](#)
- [*HAL_RTCEx_SetTamper_IT\(\)*](#)
- [*HAL_RTCEx_DeactivateTamper\(\)*](#)
- [*HAL_RTCEx_TamperTimeStampIRQHandler\(\)*](#)
- [*HAL_RTCEx_TimeStampEventCallback\(\)*](#)
- [*HAL_RTCEx_Tamper1EventCallback\(\)*](#)
- [*HAL_RTCEx_Tamper2EventCallback\(\)*](#)
- [*HAL_RTCEx_Tamper3EventCallback\(\)*](#)

- [*HAL_RTCEx_PollForTimeStampEvent\(\)*](#)
- [*HAL_RTCEx_PollForTamper1Event\(\)*](#)
- [*HAL_RTCEx_PollForTamper2Event\(\)*](#)
- [*HAL_RTCEx_PollForTamper3Event\(\)*](#)

33.2.3 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

This section contains the following APIs:

- [*HAL_RTCEx_SetWakeUpTimer\(\)*](#)
- [*HAL_RTCEx_SetWakeUpTimer_IT\(\)*](#)
- [*HAL_RTCEx_DeactivateWakeUpTimer\(\)*](#)
- [*HAL_RTCEx_GetWakeUpTimer\(\)*](#)
- [*HAL_RTCEx_WakeUpTimerIRQHandler\(\)*](#)
- [*HAL_RTCEx_WakeUpTimerEventCallback\(\)*](#)
- [*HAL_RTCEx_PollForWakeUpTimerEvent\(\)*](#)

33.2.4 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.

This section contains the following APIs:

- [*HAL_RTCEx_BKUPWrite\(\)*](#)
- [*HAL_RTCEx_BKUPRead\(\)*](#)
- [*HAL_RTCEx_SetSmoothCalib\(\)*](#)
- [*HAL_RTCEx_SetSynchroShift\(\)*](#)
- [*HAL_RTCEx_SetCalibrationOutPut\(\)*](#)
- [*HAL_RTCEx_DeactivateCalibrationOutPut\(\)*](#)
- [*HAL_RTCEx_SetRefClock\(\)*](#)
- [*HAL_RTCEx_DeactivateRefClock\(\)*](#)
- [*HAL_RTCEx_EnableBypassShadow\(\)*](#)
- [*HAL_RTCEx_DisableBypassShadow\(\)*](#)

33.2.5 HAL_RTCEx_SetTimeStamp

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_tTimeStampEdge, uint32_t RTC_TimeStampPin)</code>
---------------	--

Function Description	SetTimeStamp.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • TimeStampEdge: Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin. • RTC_TimeStampPin: specifies the RTC TimeStamp Pin. This parameter can be one of the following values: RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin.
Return values	• HAL status
Notes	<ul style="list-style-type: none"> • This API must be called before enabling the TimeStamp feature.

33.2.6 HAL_RTCEx_SetTimeStamp_IT

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)</code>
Function Description	SetTimeStamp with Interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • TimeStampEdge: Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin. • RTC_TimeStampPin: Specifies the RTC TimeStamp Pin. This parameter can be one of the following values: RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin.
Return values	• HAL status
Notes	<ul style="list-style-type: none"> • This API must be called before enabling the TimeStamp feature.

33.2.7 HAL_RTCEx_DeactivateTimeStamp

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)</code>
Function Description	Deactivate TimeStamp.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	• HAL status

33.2.8 HAL_RTCEx_GetTimeStamp

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef *</code>
---------------	--

sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)

Function Description	Get the RTC TimeStamp value.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • sTimeStamp: Pointer to Time structure • sTimeStampDate: Pointer to Date structure • Format: specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

33.2.9 HAL_RTCEx_SetTamper

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetTamper(RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Set Tamper.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • sTamper: Pointer to Tamper Structure.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • By calling this API we disable the tamper interrupt for all tampers.

33.2.10 HAL_RTCEx_SetTamper_IT

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT(RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Sets Tamper with interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTamper: Pointer to RTC Tamper.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • By calling this API we force the tamper interrupt for all tampers.

33.2.11 HAL_RTCEx_DeactivateTamper

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper(RTC_HandleTypeDef * hrtc, uint32_t Tamper)
Function Description	Deactivate Tamper.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • Tamper: Selected tamper pin. This parameter can be any combination of RTC_TAMPER_1, RTC_TAMPER_2 and RTC_TAMPER_3.
Return values	<ul style="list-style-type: none"> • HAL status

33.2.12 HAL_RTCEx_TamperTimeStampIRQHandler

Function Name	void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	Handle TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None

33.2.13 HAL_RTCEx_TimeStampEventCallback

Function Name	void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	TimeStamp callback.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None

33.2.14 HAL_RTCEx_Tamper1EventCallback

Function Name	void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None

33.2.15 HAL_RTCEx_Tamper2EventCallback

Function Name	void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 2 callback.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None

33.2.16 HAL_RTCEx_Tamper3EventCallback

Function Name	void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 3 callback.
Parameters	<ul style="list-style-type: none">• hrtc: RTC handle
Return values	<ul style="list-style-type: none">• None

33.2.17 HAL_RTCEx_PollForTimeStampEvent

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
---------------	---

Function Description	Handle TimeStamp polling request.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

33.2.18 HAL_RTCEx_PollForTamper1Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	Handle Tamper 1 Polling.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

33.2.19 HAL_RTCEx_PollForTamper2Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	Handle Tamper 2 Polling.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

33.2.20 HAL_RTCEx_PollForTamper3Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	Handle Tamper 3 Polling.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

33.2.21 HAL_RTCEx_SetWakeUpTimer

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)
Function Description	Set wake up timer.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • WakeUpCounter: Wake up counter • WakeUpClock: Wake up clock
Return values	<ul style="list-style-type: none"> • HAL status

33.2.22 HAL_RTCEx_SetWakeUpTimer_IT

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter,
---------------	---

uint32_t WakeUpClock)

Function Description	Set wake up timer with interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • WakeUpCounter: Wake up counter • WakeUpClock: Wake up clock
Return values	<ul style="list-style-type: none"> • HAL status

33.2.23 HAL_RTCEx_DeactivateWakeUpTimer

Function Name	uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)
Function Description	Deactivate wake up timer counter.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • HAL status

33.2.24 HAL_RTCEx_GetWakeUpTimer

Function Name	uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)
Function Description	Get wake up timer counter.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • Counter value

33.2.25 HAL_RTCEx_WakeUpTimerIRQHandler

Function Name	void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	Handle Wake Up Timer interrupt request.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • None

33.2.26 HAL_RTCEx_WakeUpTimerEventCallback

Function Name	void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Wake Up Timer callback.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • None

33.2.27 HAL_RTCEx_PollForWakeUpTimerEvent

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
---------------	---

Function Description	Handle Wake Up Timer Polling.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

33.2.28 HAL_RTCEx_BKUPWrite

Function Name	void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)
Function Description	Write a data in a specified RTC Backup data register.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • BackupRegister: RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 4 to specify the register. • Data: Data to be written in the specified RTC Backup data register.
Return values	<ul style="list-style-type: none"> • None

33.2.29 HAL_RTCEx_BKUPRead

Function Name	uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)
Function Description	Reads data from the specified RTC Backup data Register.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • BackupRegister: RTC Backup data Register number. This 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"> • Read value

33.2.30 HAL_RTCEx_SetSmoothCalib

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)
Function Description	Set the Smooth calibration parameters.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • SmoothCalibPeriod: Select the Smooth Calibration Period. This parameter can be can be one of the following values : RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s. RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s. RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s. • SmoothCalibPlusPulses: Select to Set or reset the CALP bit. This parameter can be one of the following values: RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulse every 2¹¹

	<p>pulses.RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added.</p> <ul style="list-style-type: none"> • SmoothCalibMinusPulsesValue: Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue mut be equal to 0.

33.2.31 HAL_RTCEx_SetSynchroShift

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)
Function Description	Configure the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • ShiftAdd1S: Select to add or not 1 second to the time calendar. This parameter can be one of the following values : RTC_SHIFTADD1S_SET: Add one second to the clock calendar.RTC_SHIFTADD1S_RESET: No effect. • ShiftSubFS: Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When REFCKON is set, firmware must not write to Shift control register.

33.2.32 HAL_RTCEx_SetCalibrationOutPut

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)
Function Description	Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • CalibOutput: : Select the Calibration output Selection . This parameter can be one of the following values: RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.
Return values	<ul style="list-style-type: none"> • HAL status

33.2.33 HAL_RTCEx_DeactivateCalibrationOutPut

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut
---------------	--

(RTC_HandleTypeDef * hrtc)

Function Description	Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • HAL status

33.2.34 HAL_RTCEx_SetRefClock

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)
Function Description	Enable the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • HAL status

33.2.35 HAL_RTCEx_DeactivateRefClock

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)
Function Description	Disable the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • HAL status

33.2.36 HAL_RTCEx_EnableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)
Function Description	Enable the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

33.2.37 HAL_RTCEx_DisableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)
Function Description	Disable the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

33.3 RTCEx Firmware driver defines

33.3.1 RTCEx

RTCEx Add 1 Second Parameter Definition

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

RTCEx Backup Registers Definition

RTC_BKP_DR0

RTC_BKP_DR1

RTC_BKP_DR2

RTC_BKP_DR3

RTC_BKP_DR4

RTC Calibration

`__HAL_RTC_CALIBRATION_OUTPUT_ENABLE`

Description:

- Enable the RTC calibration output.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_CALIBRATION_OUTPUT_DISABLE`

Description:

- Disable the calibration output.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_CLOCKREF_DETECTION_ENABLE`

Description:

- Enable the clock reference detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_CLOCKREF_DETECTION_DISABLE`

Description:

- Disable the clock reference

detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_SHIFT_GET_FLAG`

Description:

- Get the selected RTC shift operation's flag status.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC shift operation Flag is pending or not. This parameter can be:
 - `RTC_FLAG_SHPF`

Return value:

- None

RTCEx Calib Output selection Definitions

`RTC_CALIBOUTPUT_512HZ`

`RTC_CALIBOUTPUT_1HZ`

Private macros to check input parameters

`IS_RTC_OUTPUT`

`IS_RTC_BKP`

`IS_TIMESTAMP_EDGE`

`IS_RTC_TAMPER`

`IS_RTC_TIMESTAMP_PIN`

`IS_RTC_TAMPER_TRIGGER`

`IS_RTC_TAMPER_FILTER`

`IS_RTC_TAMPER_SAMPLING_FREQ`

`IS_RTC_TAMPER_PRECHARGE_DURATION`

`IS_RTC_TAMPER_TIMESTAMPON TAMPER_DETECTION`

`IS_RTC_TAMPER_PULLUP_STATE`

`IS_RTC_WAKEUP_CLOCK`

`IS_RTC_WAKEUP_COUNTER`

`IS_RTC_SMOOTH_CALIB_PERIOD`

`IS_RTC_SMOOTH_CALIB_PLUS`

`IS_RTC_SMOOTH_CALIB_MINUS`

IS_RTC_SHIFT_ADD1S
IS_RTC_SHIFT_SUBFS
IS_RTC_CALIB_OUTPUT

RTCE_x Output Selection Definition

RTC_OUTPUT_DISABLE
RTC_OUTPUT_ALARMA
RTC_OUTPUT_WAKEUP

RTCE_x Smooth calib period Definition

RTC_SMOOTHCALIB_PERIOD_32SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2 ^{exp20} RTCCLK seconds
RTC_SMOOTHCALIB_PERIOD_16SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2 ^{exp19} RTCCLK seconds
RTC_SMOOTHCALIB_PERIOD_8SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2 ^{exp18} RTCCLK seconds

RTCE_x Smooth calib Plus pulses Definition

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

RTC Tamper

_HAL_RTC_TAMPER1_ENABLE

Description:

- Enable the RTC Tamper1 input detection.

Parameters:

- _HANDLE_: specifies the RTC handle.

Return value:

- None

_HAL_RTC_TAMPER1_DISABLE

Description:

- Disable the RTC Tamper1 input detection.

Parameters:

- _HANDLE_: specifies the RTC handle.

Return value:

- None

_HAL_RTC_TAMPER2_ENABLE

Description:

- Enable the RTC Tamper2 input

detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER2_DISABLE

Description:

- Disable the RTC Tamper2 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER3_ENABLE

Description:

- Enable the RTC Tamper3 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER3_DISABLE

Description:

- Disable the RTC Tamper3 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TAMPER_ENABLE_IT

Description:

- Enable the RTC Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
 - RTC_IT_TAMP: Tamper interrupt

Return value:

- None

`__HAL_RTC_TAMPER_DISABLE_IT`

Description:

- Disable the RTC Tamper interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
 - `RTC_IT_TAMP`: Tamper 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.
- `__INTERRUPT__`: specifies the RTC Tamper interrupt to check. This parameter can be:
 - `RTC_IT_TAMP1`: Tamper1 interrupt
 - `RTC_IT_TAMP2`: Tamper2 interrupt
 - `RTC_IT_TAMP3`: Tamper3 interrupt

Return value:

- None

`__HAL_RTC_TAMPER_GET_IT_SOURCE`

Description:

- Check whether the specified RTC Tamper interrupt has been enabled or not.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Tamper interrupt source to check. This parameter can be:
 - `RTC_IT_TAMP`: Tamper interrupt

Return value:

- None

[__HAL_RTC_TAMPER_GET_FLAG](#)**Description:**

- Get the selected RTC Tamper's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag is pending or not. This parameter can be:
 - RTC_FLAG_TAMP1F
 - RTC_FLAG_TAMP2F
 - RTC_FLAG_TAMP3F

Return value:

- None

[__HAL_RTC_TAMPER_CLEAR_FLAG](#)**Description:**

- Clear the RTC Tamper's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag to clear. This parameter can be:
 - RTC_FLAG_TAMP1F
 - RTC_FLAG_TAMP2F
 - RTC_FLAG_TAMP3F

Return value:

- None

RTCEx Tamper Filter Definition

RTC_TAMPERFILTER_DISABLE	Tamper filter is disabled
RTC_TAMPERFILTER_2SAMPLE	Tamper is activated after 2 consecutive samples at the active level
RTC_TAMPERFILTER_4SAMPLE	Tamper is activated after 4 consecutive samples at the active level
RTC_TAMPERFILTER_8SAMPLE	Tamper is activated after 8 consecutive samples at the active level.

RTCEx Tamper Pins Definition

RTC_TAMPER_1
RTC_TAMPER_2
RTC_TAMPER_3

RTCEx Tamper Pin Precharge Duration Definition

RTC_TAMPERPRECHARGEDURATION_1RTCCLK	Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
-------------------------------------	---

RTC_TAMPERPRECHARGEDURATION_2RTCCLK	Tamper pins are pre-charged before sampling during 2 RTCCLK cycles
RTC_TAMPERPRECHARGEDURATION_4RTCCLK	Tamper pins are pre-charged before sampling during 4 RTCCLK cycles
RTC_TAMPERPRECHARGEDURATION_8RTCCLK	Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

RTCEx Tamper Pull UP Definition

RTC_TAMPER_PULLUP_ENABLE	Tamper pins are pre-charged before sampling
RTC_TAMPER_PULLUP_DISABLE	TimeStamp on Tamper Detection event is not saved

RTCEx Tamper Sampling Frequencies Definition

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768	Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384	Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192	Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024	Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

EXTI RTC Tamper Timestamp EXTI_HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_IT**Description:**

- Enable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_IT`**Description:**

- Disable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_EVENT`**Description:**

- Enable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_EVENT`**Description:**

- Disable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_FALLING_EDGE`**Description:**

- Enable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_FALLING_EDGE`**Description:**

- Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

<code>__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_EDGE</code>	Description: <ul style="list-style-type: none">Enable rising edge trigger on the RTC Tamper and Timestamp associated Exti line. Return value: <ul style="list-style-type: none">None.
<code>__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_EDGE</code>	Description: <ul style="list-style-type: none">Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line. Return value: <ul style="list-style-type: none">None.
<code>__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_FALLING_EDGE</code>	Description: <ul style="list-style-type: none">Enable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line. Return value: <ul style="list-style-type: none">None.
<code>__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_FALLING_EDGE</code>	Description: <ul style="list-style-type: none">Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line. Return value: <ul style="list-style-type: none">None.
<code>__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GET_FLAG</code>	Description: <ul style="list-style-type: none">Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not. Return value: <ul style="list-style-type: none">Line: Status.

[__HAL_RTC_TAMPER_TIMESTAMP_EXTI_CLEAR_FLAG](#)**Description:**

- Clear the RTC Tamper and Timestamp associated Exti line flag.

Return value:

- None.

[__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GENERATE_SW_IT](#)**Description:**

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

RTCEx TamperTimeStampOnTamperDetection Definition

`RTC_TIMESTAMPONTAMPERDETECTION_ENABLE` TimeStamp on Tamper Detection event saved

`RTC_TIMESTAMPONTAMPERDETECTION_DISABLE` TimeStamp on Tamper Detection event is not saved

RTCEx Tamper Trigger Definition

`RTC_TAMPERTRIGGER_RISINGEDGE`

`RTC_TAMPERTRIGGER_FALLINGEDGE`

`RTC_TAMPERTRIGGER_LOWLEVEL`

`RTC_TAMPERTRIGGER_HIGHLEVEL`

RTC Timestamp[__HAL_RTC_TIMESTAMP_ENABLE](#)**Description:**

- Enable the RTC TimeStamp peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_TIMESTAMP_DISABLE](#)**Description:**

- Disable the RTC TimeStamp peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_TIMESTAMP_ENABLE_IT`

Description:

- Enable the RTC TimeStamp interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be:
 - `RTC_IT_TS`: TimeStamp interrupt

Return value:

- None

`__HAL_RTC_TIMESTAMP_DISABLE_IT`

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be:
 - `RTC_IT_TS`: TimeStamp interrupt

Return value:

- None

`__HAL_RTC_TIMESTAMP_GET_IT`

Description:

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt to check. This parameter can be:
 - `RTC_IT_TS`: TimeStamp interrupt

Return value:

- None

`__HAL_RTC_TIMESTAMP_GET_IT_SOURCE`

Description:

- Check whether the specified RTC

Time Stamp interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
 - RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC TimeStamp's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
 - RTC_FLAG_TSF
 - RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Time Stamp's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag to clear. This parameter can be:
 - RTC_FLAG_TSF

Return value:

- None

RTCEx TimeStamp Pin Selection

RTC_TIMESTAMPIN_DEFAULT

RTCEx Time Stamp Edges definition

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

RTC WakeUp Timer

[__HAL_RTC_WAKEUPTIMER_ENABLE](#)**Description:**

- Enable the RTC WakeUp Timer peripheral.

Parameters:

- [__HANDLE__](#): specifies the RTC handle.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_DISABLE](#)**Description:**

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- [__HANDLE__](#): specifies the RTC handle.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_ENABLE_IT](#)**Description:**

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- [__HANDLE__](#): specifies the RTC handle.
- [__INTERRUPT__](#): specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
 - [RTC_IT_WUT](#): WakeUpTimer 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 disabled. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

Description:

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt to check. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

Description:

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

Description:

- Get the selected RTC WakeUpTimer's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
 - RTC_FLAG_WUT_F
 - RTC_FLAG_WUT_WF

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 WakeUpTimer Flag to clear. This parameter can be:
 - RTC_FLAG_WUT_F

Return value:

- None

_HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_IT**Description:**

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

Description:

- Disable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

Description:

- Enable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Disable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Enable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Disable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Enable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Disable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

Description:

- Clear the RTC WakeUp Timer associated Exti line flag.

Return value:

- None.

Description:

- Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

RTCEx Wakeup Timer Definition

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

34 HAL SMARTCARD Generic Driver

34.1 SMARTCARD Firmware driver registers structures

34.1.1 SMARTCARD_InitTypeDef

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) / ((hsmcard->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_Enable`**
- **`uint32_t SMARTCARD_InitTypeDef::TimeOutEnable`**
Specifies whether the receiver timeout is enabled. This parameter can be a value of **`SMARTCARD_Timeout_Enable`**
- **`uint32_t SMARTCARD_InitTypeDef::TimeOutValue`**
Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- **`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

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

Data Fields

- **`USART_TypeDef * Instance`**
- **`SMARTCARD_InitTypeDef Init`**
- **`SMARTCARD_AdvFeatureInitTypeDef AdvancedInit`**
- **`uint8_t * pTxBuffPtr`**
- **`uint16_t TxXferSize`**
- **`uint16_t TxXferCount`**
- **`uint8_t * pRxBuffPtr`**
- **`uint16_t RxXferSize`**
- **`uint16_t RxXferCount`**
- **`DMA_HandleTypeDef * hdmatx`**
- **`DMA_HandleTypeDef * hdmarx`**
- **`HAL_LockTypeDef Lock`**
- **`__IO HAL_SMARTCARD_StateTypeDef State`**
- **`__IO uint32_t ErrorCode`**

Field Documentation

- **`USART_TypeDef* SMARTCARD_HandleTypeDef::Instance`**
USART registers base address
- **`SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init`**
SmartCard communication parameters
- **`SMARTCARD_AdvFeatureInitTypeDef SMARTCARD_HandleTypeDef::AdvancedInit`**
SmartCard advanced features initialization parameters
- **`uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr`**
Pointer to SmartCard Tx transfer Buffer
- **`uint16_t SMARTCARD_HandleTypeDef::TxXferSize`**
SmartCard Tx Transfer size

- ***uint16_t SMARTCARD_HandleTypeDef::TxXferCount***
SmartCard Tx Transfer Counter
- ***uint8_t* SMARTCARD_HandleTypeDef::pRxBuffPtr***
Pointer to SmartCard Rx transfer Buffer
- ***uint16_t SMARTCARD_HandleTypeDef::RxXferSize***
SmartCard Rx Transfer size
- ***uint16_t SMARTCARD_HandleTypeDef::RxXferCount***
SmartCard Rx Transfer Counter
- ***DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmatx***
SmartCard Tx DMA Handle parameters
- ***DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmarx***
SmartCard Rx DMA Handle parameters
- ***HAL_LockTypeDef SMARTCARD_HandleTypeDef::Lock***
Locking object
- ***_IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::State***
SmartCard communication state
- ***_IO uint32_t SMARTCARD_HandleTypeDef::ErrorCode***
SmartCard Error code This parameter can be a value of [SMARTCARD_Error](#)

34.2 SMARTCARD Firmware driver API description

34.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure (eg. SMARTCARD_HandleTypeDef hsmartcard).
2. Associate a USART to the SMARTCARD handle hsmartcard.
3. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MsplInit() API:
 - Enable the USARTx interface clock.
 - USART pins configuration:
 - Enable the clock for the USART GPIOs.
 - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
 - 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.
4. 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 hsmcard handle Init structure.
5. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmcard handle AdvancedInit structure.
 6. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
 - This API configures also the low level Hardware (GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit() 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.

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 or not the specified SMARTCARD flag is set
- __HAL_SMARTCARD_CLEAR_FLAG : Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_GET_IT_SOURCE: Check whether or not the specified SMARTCARD interrupt is enabled



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 USARTx associated to the SmartCard.

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.

- These parameters can be configured:
 - Baud Rate
 - Parity: parity should be enabled, Frame Length is fixed to 8 bits plus parity: the SMARTCARD 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 follows the USART synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [**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.
- There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA, the relevant API's return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
 - The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected.
- Blocking mode APIs are :
 - HAL_SMARTCARD_Transmit()
 - HAL_SMARTCARD_Receive()
- Non Blocking mode APIs with Interrupt are :
 - HAL_SMARTCARD_Transmit_IT()
 - HAL_SMARTCARD_Receive_IT()
 - HAL_SMARTCARD_IRQHandler()
- Non Blocking mode functions with DMA are :

- HAL_SMARTCARD_Transmit_DMA()
- HAL_SMARTCARD_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - HAL_SMARTCARD_TxCpltCallback()
 - HAL_SMARTCARD_RxCpltCallback()
 - HAL_SMARTCARD_ErrorCallback()

This section contains the following APIs:

- [*HAL_SMARTCARD_Transmit\(\)*](#)
- [*HAL_SMARTCARD_Receive\(\)*](#)
- [*HAL_SMARTCARD_Transmit_IT\(\)*](#)
- [*HAL_SMARTCARD_Receive_IT\(\)*](#)
- [*HAL_SMARTCARD_Transmit_DMA\(\)*](#)
- [*HAL_SMARTCARD_Receive_DMA\(\)*](#)
- [*HAL_SMARTCARD_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 handle 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() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- [*HAL_SMARTCARD_GetState\(\)*](#)
- [*HAL_SMARTCARD_GetError\(\)*](#)

34.2.5 HAL_SMARTCARD_Init

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	Initialize the SMARTCARD mode according to the specified parameters in the SMARTCARD_HandleTypeDef and initialize the associated handle.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.6 HAL_SMARTCARD_DelInit

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_DelInit (SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	Delinitialize the SMARTCARD peripheral.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the

specified SMARTCARD module.

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

34.2.7 HAL_SMARTCARD_MspInit

Function Name	void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	Initialize the SMARTCARD MSP.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> • None

34.2.8 HAL_SMARTCARD_MspDelInit

Function Name	void HAL_SMARTCARD_MspDelInit (SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	DeInitialize the SMARTCARD MSP.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> • None

34.2.9 HAL_SMARTCARD_Transmit

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: pointer to data buffer. • Size: amount of data to be sent. • Timeout: : Timeout duration.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.10 HAL_SMARTCARD_Receive

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: pointer to data buffer. • Size: amount of data to be received.

- **Timeout:** : Timeout duration.

Return values • HAL status

34.2.11 HAL_SMARTCARD_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: pointer to data buffer. • Size: amount of data to be sent.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.12 HAL_SMARTCARD_Receive_IT

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: pointer to data buffer. • Size: amount of data to be received.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.13 HAL_SMARTCARD_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: pointer to data buffer. • Size: amount of data to be sent.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.14 HAL_SMARTCARD_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in DMA mode.

Parameters	<ul style="list-style-type: none"> hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. pData: pointer to data buffer. Size: amount of data to be received.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).

34.2.15 HAL_SMARTCARD_IRQHandler

Function Name	void HAL_SMARTCARD_IRQHandler(SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	Handle SMARTCARD interrupt requests.
Parameters	<ul style="list-style-type: none"> hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> None

34.2.16 HAL_SMARTCARD_TxCpltCallback

Function Name	void HAL_SMARTCARD_TxCpltCallback(SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> None

34.2.17 HAL_SMARTCARD_RxCpltCallback

Function Name	void HAL_SMARTCARD_RxCpltCallback(SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> None

34.2.18 HAL_SMARTCARD_ErrorCallback

Function Name	void HAL_SMARTCARD_ErrorCallback(SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	SMARTCARD error callback.
Parameters	<ul style="list-style-type: none"> hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

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

34.2.19 HAL_SMARTCARD_GetState

Function Name	HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	Return the SMARTCARD handle state.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> • SMARTCARD handle state

34.2.20 HAL_SMARTCARD_GetError

Function Name	uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	Return the SMARTCARD handle error code.
Parameters	<ul style="list-style-type: none"> • hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> • SMARTCARD handle Error Code

34.3 SMARTCARD Firmware driver defines

34.3.1 SMARTCARD

SMARTCARD advanced feature initialization type

SMARTCARD_ADVFEATURE_NO_INIT	No advanced feature initialization
SMARTCARD_ADVFEATURE_TXINVERT_INIT	TX pin active level inversion
SMARTCARD_ADVFEATURE_RXINVERT_INIT	RX pin active level inversion
SMARTCARD_ADVFEATURE_DATAINVERT_INIT	Binary data inversion
SMARTCARD_ADVFEATURE_SWAP_INIT	TX/RX pins swap
SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT	RX overrun disable
SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT	DMA disable on Reception Error
SMARTCARD_ADVFEATURE_MSBFIRST_INIT	Most significant bit sent/received first

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE SMARTCARD frame phase on first clock transition

`SMARTCARD_PHASE_2EDGE` SMARTCARD frame phase on second clock transition

SMARTCARD Clock Polarity

`SMARTCARD_POLARITY_LOW` SMARTCARD frame low polarity

`SMARTCARD_POLARITY_HIGH` SMARTCARD frame high polarity

SMARTCARD auto retry counter LSB position in CR3 register

`SMARTCARD_CR3_SCARCNT_LSB_POS` SMARTCARD auto retry counter LSB position in CR3 register

SMARTCARD advanced feature Binary Data inversion

`SMARTCARD_ADVFEATURE_DATAINV_DISABLE` Binary data inversion disable

`SMARTCARD_ADVFEATURE_DATAINV_ENABLE` Binary data inversion enable

SMARTCARD advanced feature DMA Disable on Rx Error

`SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR` DMA enable on Reception Error

`SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR` DMA disable on Reception Error

SMARTCARD Error

`HAL_SMARTCARD_ERROR_NONE` No error

`HAL_SMARTCARD_ERROR_PE` Parity error

`HAL_SMARTCARD_ERROR_NE` Noise error

`HAL_SMARTCARD_ERROR_FE` frame error

`HAL_SMARTCARD_ERROR_ORE` Overrun error

`HAL_SMARTCARD_ERROR_DMA` DMA transfer error

`HAL_SMARTCARD_ERROR_RTO` Receiver TimeOut error

SMARTCARD Exported Macros

<code>_HAL_SMARTCARD_RESET_HANDLE_STATE</code>	Description:
	<ul style="list-style-type: none"> • Reset SMARTCARD handle state. Parameters: <ul style="list-style-type: none"> • <code>_HANDLE_</code>: SMARTCARD handle. Return value: <ul style="list-style-type: none"> • None
<code>_HAL_SMARTCARD_FLUSH_DRREGISTER</code>	Description:
	<ul style="list-style-type: none"> • Flush the Smartcard Data registers. Parameters: <ul style="list-style-type: none"> • <code>_HANDLE_</code>: specifies the SMARTCARD Handle. Return value: <ul style="list-style-type: none"> • None

[__HAL_SMARTCARD_CLEAR_FLAG](#)**Description:**

- Clear the specified SMARTCARD pending flag.

Parameters:

- [__HANDLE__](#): specifies the SMARTCARD Handle.
- [__FLAG__](#): specifies the flag to check. This parameter can be any combination of the following values:
 - 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_IDLEF: Idle line detected clear flag
 - SMARTCARD_CLEAR_TCF: Transmission complete clear flag
 - SMARTCARD_CLEAR_RTOF: Receiver timeout clear flag
 - SMARTCARD_CLEAR_EOBF: End of block clear flag

Return value:

- None

[__HAL_SMARTCARD_CLEAR_PEFLAG](#)**Description:**

- Clear the SMARTCARD PE pending flag.

Parameters:

- [__HANDLE__](#): specifies the SMARTCARD Handle.

Return value:

- None

[__HAL_SMARTCARD_CLEAR_FEFLAG](#)**Description:**

- Clear the SMARTCARD FE pending flag.

Parameters:

- [__HANDLE__](#): specifies the SMARTCARD Handle.

Return value:

- None

[__HAL_SMARTCARD_CLEAR_NEFLAG](#)**Description:**

- Clear the SMARTCARD NE pending flag.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

`__HAL_SMARTCARD_CLEAR_OREFLAG`

G

Description:

- Clear the SMARTCARD ORE pending flag.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

`__HAL_SMARTCARD_CLEAR_IDLEFLAG`

G

Description:

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

`__HAL_SMARTCARD_GET_FLAG`

Description:

- Check 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: Transfer complete flag

- Transmission complete flag
- SMARTCARD_FLAG_RXNE: Receive data register not empty flag
- SMARTCARD_FLAG_IDLE: Idle line detection flag
- SMARTCARD_FLAG_ORE: Overrun error flag
- SMARTCARD_FLAG_NE: Noise error flag
- SMARTCARD_FLAG_FE: Framing error flag
- SMARTCARD_FLAG_PE: Parity error flag

Return value:

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

__HAL_SMARTCARD_ENABLE_IT**Description:**

- Enable 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_EOB: End of block interrupt
 - SMARTCARD_IT_RTO: 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_IDLE: Idle line detection interrupt
 - SMARTCARD_IT_PE: Parity error interrupt
 - SMARTCARD_IT_ERR: Error interrupt(frame error, noise error, overrun error)

Return value:

- None

__HAL_SMARTCARD_DISABLE_IT**Description:**

- Disable the specified SmartCard interrupt.

Parameters:

- `_HANDLE_`: specifies the SMARTCARD Handle. 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_EOB: End of block interrupt
 - SMARTCARD_IT_RTO: 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_IDLE: Idle line detection interrupt
 - SMARTCARD_IT_PE: Parity error interrupt
 - SMARTCARD_IT_ERR: Error interrupt(frame error, noise error, overrun error)

Return value:

- None

`_HAL_SMARTCARD_GET_IT`**Description:**

- Check 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_EOB: End of block interrupt
 - SMARTCARD_IT_RTO: 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_IDLE: Idle line detection 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`
E

Description:

- Check whether the specified SmartCard interrupt source is enabled 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 source to check. This parameter can be one of the following values:
 - SMARTCARD_IT_EOB: End of block interrupt
 - SMARTCARD_IT_RTO: 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_IDLE: Idle line detection 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:

- Clear 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:
 - 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_IDLEF: Idle line detection clear flag
 - SMARTCARD_CLEAR_TCF: Transmission complete clear flag
 - SMARTCARD_CLEAR_RTOF: Receiver timeout clear flag
 - SMARTCARD_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_R EQUEST: Receive data flush Request
 - SMARTCARD_TXDATA_FLUSH_R EQUEST: Transmit data flush Request

Return value:

- None

`_HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE`**Description:**

- Enable the SMARTCARD one bit sample

method.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

Description:

- Disable the SMARTCARD one bit sample method.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

Description:

- 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

Description:

- 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

SMARTCARDEX Exported Macros

`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

<code>SMARTCARD_FLAG_RXACK</code>	SMARTCARD receive enable acknowledge flag
<code>SMARTCARD_FLAG_TEACK</code>	SMARTCARD transmit enable acknowledge flag
<code>SMARTCARD_FLAG_BUSY</code>	SMARTCARD busy flag
<code>SMARTCARD_FLAG_EOBF</code>	SMARTCARD end of block flag
<code>SMARTCARD_FLAG_RTOF</code>	SMARTCARD receiver timeout flag
<code>SMARTCARD_FLAG_TXE</code>	SMARTCARD transmit data register empty
<code>SMARTCARD_FLAG_TC</code>	SMARTCARD transmission complete
<code>SMARTCARD_FLAG_RXNE</code>	SMARTCARD read data register not empty
<code>SMARTCARD_FLAG_IDLE</code>	SMARTCARD idle line detection
<code>SMARTCARD_FLAG_ORE</code>	SMARTCARD overrun error
<code>SMARTCARD_FLAG_NE</code>	SMARTCARD noise error
<code>SMARTCARD_FLAG_FE</code>	SMARTCARD frame error
<code>SMARTCARD_FLAG_PE</code>	SMARTCARD parity error

SMARTCARD guard time value LSB position in GTPR register

<code>SMARTCARD_GTPR_GT_LSB_POS</code>	SMARTCARD guard time value LSB position in GTPR register
--	--

SMARTCARD interruptions flags mask

<code>SMARTCARD_IT_MASK</code>	SMARTCARD interruptions flags mask
--------------------------------	------------------------------------

SMARTCARD Interrupts Definition

<code>SMARTCARD_IT_PE</code>	SMARTCARD parity error interruption
<code>SMARTCARD_IT_TXE</code>	SMARTCARD transmit data register empty interruption
<code>SMARTCARD_IT_TC</code>	SMARTCARD transmission complete interruption
<code>SMARTCARD_IT_RXNE</code>	SMARTCARD read data register not empty interruption
<code>SMARTCARD_IT_IDLE</code>	SMARTCARD idle line detection interruption
<code>SMARTCARD_IT_ERR</code>	SMARTCARD error interruption
<code>SMARTCARD_IT_ORE</code>	SMARTCARD overrun error interruption
<code>SMARTCARD_IT_NE</code>	SMARTCARD noise error interruption
<code>SMARTCARD_IT_FE</code>	SMARTCARD frame error interruption
<code>SMARTCARD_IT_EOB</code>	SMARTCARD end of block interruption
<code>SMARTCARD_IT_RTO</code>	SMARTCARD receiver timeout interruption

SMARTCARD Interruption Clear Flags

<code>SMARTCARD_CLEAR_PEF</code>	SMARTCARD parity error clear flag
----------------------------------	-----------------------------------

SMARTCARD_CLEAR_FEF	SMARTCARD framing error clear flag
SMARTCARD_CLEAR_NEF	SMARTCARD noise detected clear flag
SMARTCARD_CLEAR_OREF	SMARTCARD overrun error clear flag
SMARTCARD_CLEAR_IDLEF	SMARTCARD idle line detected clear flag
SMARTCARD_CLEAR_TCF	SMARTCARD transmission complete clear flag
SMARTCARD_CLEAR_RTOF	SMARTCARD receiver time out clear flag
SMARTCARD_CLEAR_EOBF	SMARTCARD end of block clear flag
SMARTCARD Last Bit	
SMARTCARD_LASTBIT_DISABLE	SMARTCARD frame last data bit clock pulse not output to SCLK pin
SMARTCARD_LASTBIT_ENABLE	SMARTCARD frame last data bit clock pulse output to SCLK pin
SMARTCARD Transfer Mode	
SMARTCARD_MODE_RX	SMARTCARD RX mode
SMARTCARD_MODE_TX	SMARTCARD TX mode
SMARTCARD_MODE_RX_RX	SMARTCARD RX and TX mode
SMARTCARD advanced feature MSB first	
SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE	Most significant bit sent/received first disable
SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE	Most significant bit sent/received first enable
SMARTCARD NACK Enable	
SMARTCARD_NACK_ENABLE	SMARTCARD NACK transmission disabled
SMARTCARD_NACK_DISABLE	SMARTCARD NACK transmission enabled
SMARTCARD One Bit Sampling Method	
SMARTCARD_ONE_BIT_SAMPLE_DISABLE	SMARTCARD frame one-bit sample disabled
SMARTCARD_ONE_BIT_SAMPLE_ENABLE	SMARTCARD frame one-bit sample enabled
SMARTCARD advanced feature Overrun Disable	
SMARTCARD_ADVFEATURE_OVERRUN_ENABLE	RX overrun enable
SMARTCARD_ADVFEATURE_OVERRUN_DISABLE	RX overrun disable
SMARTCARD Parity	
SMARTCARD_PARITY EVEN	SMARTCARD frame even parity
SMARTCARD_PARITY ODD	SMARTCARD frame odd parity
SMARTCARD Request Parameters	
SMARTCARD_RXDATA_FLUSH_REQUEST	Receive Data flush Request
SMARTCARD_TXDATA_FLUSH_REQUEST	Transmit data flush Request
SMARTCARD block length LSB position in RTOR register	

SMARTCARD_RTOR_BLEN_LSB_POS SMARTCARD block length LSB position in RTOR register

SMARTCARD advanced feature RX pin active level inversion

SMARTCARD_ADVFEATURE_RXINV_DISABLE RX pin active level inversion disable

SMARTCARD_ADVFEATURE_RXINV_ENABLE RX pin active level inversion enable

SMARTCARD advanced feature RX TX pins swap

SMARTCARD_ADVFEATURE_SWAP_DISABLE TX/RX pins swap disable

SMARTCARD_ADVFEATURE_SWAP_ENABLE TX/RX pins swap enable

SMARTCARD Stop Bits

SMARTCARD_STOPBITS_1_5 SMARTCARD frame with 1.5 stop bits

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DISABLE SMARTCARD receiver timeout disabled

SMARTCARD_TIMEOUT_ENABLE SMARTCARD receiver timeout enabled

SMARTCARD advanced feature TX pin active level inversion

SMARTCARD_ADVFEATURE_TXINV_DISABLE TX pin active level inversion disable

SMARTCARD_ADVFEATURE_TXINV_ENABLE TX pin active level inversion enable

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B SMARTCARD frame length

35 HAL SMARTCARD Extension Driver

35.1 SMARTCARDEX Firmware driver API description

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

This section contains the following APIs:

- `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(SMARTCARD_HandleTypeDef * 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">• hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.• BlockLength: SMARTCARD block length (8-bit long at most)
Return values	<ul style="list-style-type: none">• None

35.1.3 `HAL_SMARTCARDEX_TimeOut_Config`

Function Name	<code>void HAL_SMARTCARDEX_TimeOut_Config(SMARTCARD_HandleTypeDef * hsmartcard, uint32_t TimeOutValue)</code>
Function Description	Update on the fly the receiver timeout value in RTOR register.
Parameters	<ul style="list-style-type: none">• hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.• TimeOutValue: receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0xFFFFFFFF.
Return values	<ul style="list-style-type: none">• None

35.1.4 HAL_SMARTCARDEX_EnableReceiverTimeOut

Function Name	HAL_StatusTypeDef HAL_SMARTCARDEX_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	Enable the SMARTCARD receiver timeout feature.
Parameters	<ul style="list-style-type: none">• hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none">• HAL status

35.1.5 HAL_SMARTCARDEX_DisableReceiverTimeOut

Function Name	HAL_StatusTypeDef HAL_SMARTCARDEX_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)
Function Description	Disable the SMARTCARD receiver timeout feature.
Parameters	<ul style="list-style-type: none">• hsmartcard: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none">• HAL status

36 HAL SMBUS Generic Driver

36.1 SMBUS Firmware driver registers structures

36.1.1 SMBUS_InitTypeDef

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 Periphal 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

Data Fields

- ***I2C_TypeDef * Instance***
- ***SMBUS_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t 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 uint32_t SMBUS_HandleTypeDef::PreviousState***
SMBUS communication Previous state
- ***HAL_LockTypeDef SMBUS_HandleTypeDef::Lock***
SMBUS locking object
- ***__IO uint32_t SMBUS_HandleTypeDef::State***
SMBUS communication state
- ***__IO uint32_t SMBUS_HandleTypeDef::ErrorCode***
SMBUS Error code

36.2 SMBUS Firmware driver API description

36.2.1 How to use this driver

The SMBUS HAL driver can be used as follows:

1. Declare a SMBUS_HandleTypeDef handle structure, for example:
SMBUS_HandleTypeDef hsmbus;
2. Initialize the SMBUS low level resources by implementing the HAL_SMBUS_MspInit() API:
 - Enable the SMBUSx interface clock with __HAL_RCC_I2Cx_CLK_ENABLE()
 - SMBUS pins configuration
 - Enable the clock for the SMBUS GPIOs
 - Configure SMBUS pins as alternate function open-drain
 - NVIC configuration if you need to use interrupt process
 - Configure the SMBUSx interrupt priority
 - Enable the NVIC SMBUS IRQ Channel
3. Configure the Communication Clock Timing, Bus Timeout, Own Address1, Master Addressing Mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call, Nostretch mode, Peripheral mode and Packet Error Check mode in the hsmbus Init structure.
4. Initialize the SMBUS registers by calling the HAL_SMBUS_Init() API:
 - These API's configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMBUS_MspInit(&hsmbus) API.
5. To check if target device is ready for communication, use the function HAL_SMBUS_IsDeviceReady()
6. For SMBUS IO operations, only one mode of operations is available within this driver:

Interrupt mode IO operation

- Transmit in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Transmit_IT()
 - At transmission end of transfer HAL_SMBUS_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_MasterTxCpltCallback()
- Receive in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Receive_IT()
 - At reception end of transfer HAL_SMBUS_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_MasterRxCpltCallback()
- Abort a master/host SMBUS process communication with Interrupt using HAL_SMBUS_Master_Abort_IT()
 - The associated previous transfer callback is called at the end of abort process
 - mean HAL_SMBUS_MasterTxCpltCallback() in case of previous state was master transmit
 - mean HAL_SMBUS_MasterRxCpltCallback() in case of previous state was master receive
- Enable/disable the Address listen mode in slave/device or host/slave SMBUS mode using HAL_SMBUS_EnableListen_IT() HAL_SMBUS_DisableListen_IT()
 - When address slave/device SMBUS match, HAL_SMBUS_AddrCallback() is executed and user can add his own code to check the Address Match Code and the transmission direction request by master/host (Write/Read).

- At Listen mode end HAL_SMBUS_ListenCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_ListenCpltCallback()
- Transmit in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Transmit_IT()
 - At transmission end of transfer HAL_SMBUS_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_SlaveTxCpltCallback()
- Receive in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Receive_IT()
 - At reception end of transfer HAL_SMBUS_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_SlaveRxCpltCallback()
- Enable/Disable the SMBUS alert mode using HAL_SMBUS_EnableAlert_IT()
 - When SMBUS Alert is generated HAL_SMBUS_ErrorCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Alert Error Code using function HAL_SMBUS_GetError()
- Get HAL state machine or error values using HAL_SMBUS_GetState() or HAL_SMBUS_GetError()
- In case of transfer Error, HAL_SMBUS_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Error Code using function HAL_SMBUS_GetError()

SMBUS HAL driver macros list

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

- __HAL_SMBUS_ENABLE: Enable the SMBUS peripheral
- __HAL_SMBUS_DISABLE: Disable the SMBUS peripheral
- __HAL_SMBUS_GET_FLAG : Checks whether the specified SMBUS flag is set or not
- __HAL_SMBUS_CLEAR_FLAG : Clears the specified SMBUS pending flag
- __HAL_SMBUS_ENABLE_IT: Enables the specified SMBUS interrupt
- __HAL_SMBUS_DISABLE_IT: Disables the specified SMBUS interrupt



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

36.2.2 Initialization and de-initialization functions

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

- User must Implement HAL_SMBUS_MsplInit() 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.

This section contains the following APIs:

- [***HAL_SMBUS_Init\(\)***](#)
- [***HAL_SMBUS_DelInit\(\)***](#)
- [***HAL_SMBUS_MspInit\(\)***](#)
- [***HAL_SMBUS_MspDelInit\(\)***](#)

36.2.3 IO operation functions

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

1. Blocking mode function to check if device is ready for usage is :
 - `HAL_SMBUS_IsDeviceReady()`
2. There is only one mode of transfer:
 - 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_EnableListen_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_AddrCallback()`
 - `HAL_SMBUS_ListenCpltCallback()`
 - `HAL_SMBUS_ErrorCallback()`

This section contains the following APIs:

- [***HAL_SMBUS_Master_Transmit_IT\(\)***](#)
- [***HAL_SMBUS_Master_Receive_IT\(\)***](#)
- [***HAL_SMBUS_Master_Abort_IT\(\)***](#)
- [***HAL_SMBUS_Slave_Transmit_IT\(\)***](#)
- [***HAL_SMBUS_Slave_Receive_IT\(\)***](#)
- [***HAL_SMBUS_EnableListen_IT\(\)***](#)
- [***HAL_SMBUS_DisableListen_IT\(\)***](#)
- [***HAL_SMBUS_EnableAlert_IT\(\)***](#)

- `HAL_SMBUS_DisableAlert_IT()`
- `HAL_SMBUS_IsDeviceReady()`

36.2.4 Peripheral State and Errors functions

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

This section contains the following APIs:

- `HAL_SMBUS_GetState()`
- `HAL_SMBUS_GetError()`

36.2.5 HAL_SMBUS_Init

Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_Init(SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	Initialize the SMBUS according to the specified parameters in the <code>SMBUS_InitTypeDef</code> and initialize the associated handle.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a <code>SMBUS_HandleTypeDef</code> structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • HAL status

36.2.6 HAL_SMBUS_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_DelInit(SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	DeInitialize the SMBUS peripheral.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a <code>SMBUS_HandleTypeDef</code> structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • HAL status

36.2.7 HAL_SMBUS_MspInit

Function Name	<code>void HAL_SMBUS_MspInit(SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	Initialize the SMBUS MSP.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a <code>SMBUS_HandleTypeDef</code> structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • None

36.2.8 HAL_SMBUS_MspDelInit

Function Name	<code>void HAL_SMBUS_MspDelInit(SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	DeInitialize the SMBUS MSP.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a <code>SMBUS_HandleTypeDef</code> structure that contains the configuration information for the specified

SMBUS.

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

36.2.9 HAL_SMBUS_Master_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)
Function Description	Transmit in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. DevAddress: Target device address pData: Pointer to data buffer Size: Amount of data to be sent XferOptions: Options of Transfer, value of SMBUS XferOptions definition
Return values	<ul style="list-style-type: none"> HAL status

36.2.10 HAL_SMBUS_Master_Receive_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)
Function Description	Receive in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. DevAddress: Target device address pData: Pointer to data buffer Size: Amount of data to be sent XferOptions: Options of Transfer, value of SMBUS XferOptions definition
Return values	<ul style="list-style-type: none"> HAL status

36.2.11 HAL_SMBUS_Master_Abort_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)
Function Description	Abort a master/host SMBUS process communication with Interrupt.
Parameters	<ul style="list-style-type: none"> hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. DevAddress: Target device address
Return values	<ul style="list-style-type: none"> HAL status

Notes	<ul style="list-style-type: none"> This abort can be called only if state is ready
36.2.12 HAL_SMBUS_Slave_Transmit_IT	
Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)</code>
Function Description	Transmit in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. pData: Pointer to data buffer Size: Amount of data to be sent XferOptions: Options of Transfer, value of SMBUS XferOptions definition
Return values	<ul style="list-style-type: none"> HAL status
36.2.13 HAL_SMBUS_Slave_Receive_IT	
Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)</code>
Function Description	Receive in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. pData: Pointer to data buffer Size: Amount of data to be sent XferOptions: Options of Transfer, value of SMBUS XferOptions definition
Return values	<ul style="list-style-type: none"> HAL status
36.2.14 HAL_SMBUS_EnableListen_IT	
Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	Enable the Address listen mode with Interrupt.
Parameters	<ul style="list-style-type: none"> hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> HAL status
36.2.15 HAL_SMBUS_DisableListen_IT	
Function Name	<code>HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)</code>
Function Description	Disable the Address listen mode with Interrupt.

Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • HAL status

36.2.16 HAL_SMBUS_EnableAlert_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)
Function Description	Enable the SMBUS alert mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hsmbus: : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	<ul style="list-style-type: none"> • HAL status

36.2.17 HAL_SMBUS_DisableAlert_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)
Function Description	Disable the SMBUS alert mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hsmbus: : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	<ul style="list-style-type: none"> • HAL status

36.2.18 HAL_SMBUS_IsDeviceReady

Function Name	HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)
Function Description	Check if target device is ready for communication.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • DevAddress: Target device address • Trials: Number of trials • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- This function is used with Memory devices

36.2.19 HAL_SMBUS_EV_IRQHandler

Function Name	void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)
Function Description	Handle SMBUS event interrupt request.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified

SMBUS.

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

36.2.20 HAL_SMBUS_ER_IRQHandler

Function Name	void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)
Function Description	Handle SMBUS error interrupt request.
Parameters	<ul style="list-style-type: none"> hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> None

36.2.21 HAL_SMBUS_MasterTxCpltCallback

Function Name	void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Master Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> None

36.2.22 HAL_SMBUS_MasterRxCpltCallback

Function Name	void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Master Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> None

36.2.23 HAL_SMBUS_SlaveTxCpltCallback

Function Name	void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Slave Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> None

36.2.24 HAL_SMBUS_SlaveRxCpltCallback

Function Name	void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
---------------	--

Function Description	Slave Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • None

36.2.25 HAL_SMBUS_AddrCallback

Function Name	void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)
Function Description	Slave Address Match callback.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • TransferDirection: Master request Transfer Direction (Write/Read) • AddrMatchCode: Address Match Code
Return values	<ul style="list-style-type: none"> • None

36.2.26 HAL_SMBUS_ListenCpltCallback

Function Name	void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Listen Complete callback.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • None

36.2.27 HAL_SMBUS_ErrorCallback

Function Name	void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	SMBUS error callback.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • None

36.2.28 HAL_SMBUS_GetState

Function Name	uint32_t HAL_SMBUS_GetState (SMBUS_HandleTypeDef * hsmbus)
Function Description	Return the SMBUS handle state.
Parameters	<ul style="list-style-type: none"> • hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values	<ul style="list-style-type: none"> • HAL state
---------------	---

36.2.29 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"> • hsmbus: : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	<ul style="list-style-type: none"> • SMBUS Error Code

36.3 SMBUS Firmware driver defines

36.3.1 SMBUS

SMBUS addressing mode

`SMBUS_ADDRESSINGMODE_7BIT`

`SMBUS_ADDRESSINGMODE_10BIT`

SMBUS Analog Filter

`SMBUS_ANALOGFILTER_ENABLE`

`SMBUS_ANALOGFILTER_DISABLE`

SMBUS dual addressing mode

`SMBUS_DUALADDRESS_DISABLE`

`SMBUS_DUALADDRESS_ENABLE`

SMBUS Error Code definition

<code>HAL_SMBUS_ERROR_NONE</code>	No error
-----------------------------------	----------

<code>HAL_SMBUS_ERROR_BERR</code>	BERR error
-----------------------------------	------------

<code>HAL_SMBUS_ERROR_ARLO</code>	ARLO error
-----------------------------------	------------

<code>HAL_SMBUS_ERROR_ACKF</code>	ACKF error
-----------------------------------	------------

<code>HAL_SMBUS_ERROR_OVR</code>	OVR error
----------------------------------	-----------

<code>HAL_SMBUS_ERROR_HALTIMEOUT</code>	Timeout error
---	---------------

<code>HAL_SMBUS_ERROR_BUSTIMEOUT</code>	Bus Timeout error
---	-------------------

<code>HAL_SMBUS_ERROR_ALERT</code>	Alert error
------------------------------------	-------------

<code>HAL_SMBUS_ERROR_PECERR</code>	PEC error
-------------------------------------	-----------

SMBUS Exported Macros

`_HAL_SMBUS_RESET_HANDLE_STATE` **Description:**

- Reset SMBUS handle state.

Parameters:

- `_HANDLE_`: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_ENABLE_IT

- Enable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to enable. 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

- Disable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to 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_GET_IT_SOURCE](#)**Description:**

- Check whether the specified SMBUS interrupt source is enabled or not.

Parameters:

- [__HANDLE__](#): specifies the SMBUS Handle.
- [__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
 - 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:**

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

Parameters:

- [__HANDLE__](#): specifies the SMBUS Handle.
- [__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:

- Clear the SMBUS pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.
- `__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`

Description:

- Enable the specified SMBUS peripheral.

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.

Return value:

- None

`__HAL_SMBUS_DISABLE`

Description:

- Disable the specified SMBUS peripheral.

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.

Return value:

- None

`__HAL_SMBUS_GENERATE_NACK`

Description:

- Generate a Non-Acknowledge SMBUS peripheral in Slave mode.

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.

Return value:

- None

SMBUS Flag definition

`SMBUS_FLAG_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_DISABLE
SMBUS_GENERALCALL_ENABLE

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_DISABLE
SMBUS_NOSTRETCH_ENABLE

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

SMBUS packet error check mode

SMBUS_PEC_DISABLE

SMBUS_PEC_ENABLE
SMBUS peripheral mode
SMBUS_PERIPHERAL_MODE_SMBUS_HOST
SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE
SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP
SMBUS ReloadEndMode definition
SMBUS_SOFTEND_MODE
SMBUS_RELOAD_MODE
SMBUS_AUTOEND_MODE
SMBUS_SENDPEC_MODE
SMBUS StartStopMode definition
SMBUS_NO_STARTSTOP
SMBUS_GENERATE_STOP
SMBUS_GENERATE_START_READ
SMBUS_GENERATE_START_WRITE
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

37 HAL SPI Generic Driver

37.1 SPI Firmware driver registers structures

37.1.1 SPI_InitTypeDef

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_TIMode**
- ***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 can be a value of **SPI_CRC_length**
- ***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

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***
- ***__IO HAL_SPI_StateTypeDef State***
- ***uint32_t 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`
- `_IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State`
- `uint32_t __SPI_HandleTypeDef::ErrorCode`

37.2 SPI Firmware driver API description

37.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit() API:
 - a. Enable the SPIx interface clock
 - b. SPI pins configuration
 - Enable the clock for the SPI GPIOs
 - Configure these SPI pins as alternate function push-pull
 - c. NVIC configuration if you need to use interrupt process
 - Configure the SPIx interrupt priority
 - Enable the NVIC SPI IRQ handle
 - d. DMA Configuration if you need to use DMA process
 - Declare a DMA_HandleTypeDef handle structure for the transmit or receive channel
 - Enable the DMAx clock
 - Configure the DMA handle parameters
 - Configure the DMA Tx or Rx channel
 - Associate the initialized hdma_tx 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:
 - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit() API.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
 - a. Master 2Lines RxOnly
 - b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled
3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL_SPI_DMAPause() / HAL_SPI_DMAStop() only under the SPI callbacks

Using the HAL it is not possible to reach all supported SPI frequency with the different frequencies the following table resume the max SPI frequency reached with 8-bit or 16-bit data, according to frequency used on APBx Peripheral Clock (fPCLK) used by the SPI instance:

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	$f_{CPU}/8$	$f_{CPU}/16$



The max SPI frequency depend on SPI data size (4bits, 5bits,..., 8bits,...15bits, 16 SPI mode(2 Lines fullduplex, 2 lines RxOnly, 1 line TX/RX) and Process mode (Polling, IT, DMA)

1. TX/RX processes are HAL_SPI_TransmitReceive(), HAL_SPI_TransmitReceive_IT() and HAL_SPI_TransmitReceive_IT()
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 de-initialization functions

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

- User must implement HAL_SPI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPI_Init() to configure the selected device with the selected configuration:
 - Mode
 - Direction
 - Data Size
 - Clock Polarity and Phase
 - NSS Management
 - BaudRate Prescaler
 - FirstBit
 - TIMode
 - CRC Calculation
 - CRC Polynomial if CRC enabled
 - CRC Length, used only with Data8 and Data16
 - FIFO reception threshold
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

- [HAL_SPI_Init\(\)](#)

- [*HAL_SPI_DelInit\(\)*](#)
- [*HAL_SPI_MspInit\(\)*](#)
- [*HAL_SPI_MspDelInit\(\)*](#)

37.2.3 IO operation functions

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

The SPI supports master and slave mode :

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The *HAL_SPI_TxCpltCallback()*, *HAL_SPI_RxCpltCallback()* and *HAL_SPI_TxRxCpltCallback()* user callbacks will be executed respectively at the end of the transmit or Receive process The *HAL_SPI_ErrorCallback()* user callback will be executed when a communication error is detected
2. APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.

This section contains the following APIs:

- [*HAL_SPI_Transmit\(\)*](#)
- [*HAL_SPI_Receive\(\)*](#)
- [*HAL_SPI_TransmitReceive\(\)*](#)
- [*HAL_SPI_Transmit_IT\(\)*](#)
- [*HAL_SPI_Receive_IT\(\)*](#)
- [*HAL_SPI_TransmitReceive_IT\(\)*](#)
- [*HAL_SPI_Transmit_DMA\(\)*](#)
- [*HAL_SPI_Receive_DMA\(\)*](#)
- [*HAL_SPI_TransmitReceive_DMA\(\)*](#)
- [*HAL_SPI_DMAPause\(\)*](#)
- [*HAL_SPI_DMAResume\(\)*](#)
- [*HAL_SPI_DMAStop\(\)*](#)
- [*HAL_SPI_IRQHandler\(\)*](#)
- [*HAL_SPI_TxCpltCallback\(\)*](#)
- [*HAL_SPI_RxCpltCallback\(\)*](#)
- [*HAL_SPI_TxRxCpltCallback\(\)*](#)
- [*HAL_SPI_TxHalfCpltCallback\(\)*](#)
- [*HAL_SPI_RxHalfCpltCallback\(\)*](#)
- [*HAL_SPI_TxRxHalfCpltCallback\(\)*](#)
- [*HAL_SPI_ErrorCallback\(\)*](#)

37.2.4 Peripheral State and Errors functions

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

- *HAL_SPI_GetState()* API can be helpful to check in run-time the state of the SPI peripheral
- *HAL_SPI_GetError()* check in run-time Errors occurring during communication

This section contains the following APIs:

- `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	Initialize the SPI according to the specified parameters in the SPI_InitTypeDef and initialize the associated handle.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • HAL status

37.2.6 HAL_SPI_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)</code>
Function Description	Deinitialize the SPI peripheral.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • HAL status

37.2.7 HAL_SPI_MspInit

Function Name	<code>void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)</code>
Function Description	Initialize the SPI MSP.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

37.2.8 HAL_SPI_MspDeInit

Function Name	<code>void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)</code>
Function Description	Deinitialize the SPI MSP.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

37.2.9 HAL_SPI_Transmit

Function Name	<code>HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)</code>
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent • Timeout: Timeout duration

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

37.2.10 HAL_SPI_Receive

Function Name	HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be received • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

37.2.11 HAL_SPI_TransmitReceive

Function Name	HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Transmit and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pTxData: pointer to transmission data buffer • pRxData: pointer to reception data buffer • Size: amount of data to be sent and received • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

37.2.12 HAL_SPI_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

37.2.13 HAL_SPI_Receive_IT

Function Name	HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent

Return values	<ul style="list-style-type: none"> • HAL status
37.2.14 HAL_SPI_TransmitReceive_IT	
Function Name	HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Transmit and Receive an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pTxData: pointer to transmission data buffer • pRxData: pointer to reception data buffer • Size: amount of data to be sent and received
Return values	<ul style="list-style-type: none"> • HAL status
37.2.15 HAL_SPI_Transmit_DMA	
Function Name	HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
37.2.16 HAL_SPI_Receive_DMA	
Function Name	HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> When the CRC feature is enabled the pData Length must be Size + 1.
37.2.17 HAL_SPI_TransmitReceive_DMA	
Function Name	HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Transmit and Receive an amount of data in non-blocking mode with DMA.

Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. pTxData: pointer to transmission data buffer pRxData: pointer to reception data buffer Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When the CRC feature is enabled the pRxData Length must be Size + 1

37.2.18 HAL_SPI_DMAPause

Function Name	HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)
Function Description	Pause the DMA Transfer.
Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none"> HAL status

37.2.19 HAL_SPI_DMAResume

Function Name	HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)
Function Description	Resume the DMA Transfer.
Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none"> HAL status

37.2.20 HAL_SPI_DMAStop

Function Name	HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)
Function Description	Stop the DMA Transfer.
Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none"> HAL status

37.2.21 HAL_SPI_IRQHandler

Function Name	void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)
Function Description	Handle SPI interrupt request.
Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none"> None

37.2.22 HAL_SPI_TxCpltCallback

Function Name	void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

37.2.23 HAL_SPI_RxCpltCallback

Function Name	void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

37.2.24 HAL_SPI_TxRxCpltCallback

Function Name	void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx and Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

37.2.25 HAL_SPI_TxHalfCpltCallback

Function Name	void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

37.2.26 HAL_SPI_RxHalfCpltCallback

Function Name	void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Rx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

37.2.27 HAL_SPI_TxRxHalfCpltCallback

Function Name	void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx and Rx Half Transfer callback.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains

the configuration information for SPI module.

Return values • None

37.2.28 HAL_SPI_ErrorCallback

Function Name **void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)**

Function Description SPI error callback.

Parameters • **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values • None

37.2.29 HAL_SPI_GetState

Function Name **HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)**

Function Description Return the SPI handle state.

Parameters • **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values • SPI state

37.2.30 HAL_SPI_GetError

Function Name **uint32_t HAL_SPI_GetError (SPI_HandleTypeDef * hspi)**

Function Description Return the SPI error code.

Parameters • **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values • SPI error code in bitmap format

37.3 SPI Firmware driver defines

37.3.1 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 1EDGE

SPI_PHASE_2EDGE SPI Phase 2EDGE

IS_SPI_CPHA

SPI Clock Polarity

SPI_POLARITY_LOW SPI polarity Low

SPI_POLARITY_HIGH SPI polarity High

IS_SPI_CPOL

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

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 = 4bits

SPI_DATASIZE_5BIT SPI Datasize = 5bits

SPI_DATASIZE_6BIT SPI Datasize = 6bits

SPI_DATASIZE_7BIT SPI Datasize = 7bits

SPI_DATASIZE_8BIT SPI Datasize = 8bits

SPI_DATASIZE_9BIT SPI Datasize = 9bits

SPI_DATASIZE_10BIT SPI Datasize = 10bits

SPI_DATASIZE_11BIT SPI Datasize = 11bits

SPI_DATASIZE_12BIT SPI Datasize = 12bits

SPI_DATASIZE_13BIT SPI Datasize = 13bits

SPI_DATASIZE_14BIT SPI Datasize = 14bits

SPI_DATASIZE_15BIT SPI Datasize = 15bits

SPI_DATASIZE_16BIT SPI Datasize = 16bits

IS_SPI_DATASIZE

SPI Direction Mode

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

IS_SPI_DIRECTION

`IS_SPI_DIRECTION_2LINES`
`IS_SPI_DIRECTION_2LINES_OR_1LINE`

SPI Error Code

<code>HAL_SPI_ERROR_NONE</code>	No error
<code>HAL_SPI_ERROR_MODF</code>	MODF error
<code>HAL_SPI_ERROR_CRC</code>	CRC error
<code>HAL_SPI_ERROR_OVR</code>	OVR error
<code>HAL_SPI_ERROR_FRE</code>	FRE error
<code>HAL_SPI_ERROR_DMA</code>	DMA transfer error
<code>HAL_SPI_ERROR_FLAG</code>	Error on BSY/TXE/FTLVL/FRLVL Flag
<code>HAL_SPI_ERROR_UNKNOW</code>	Unknown error

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:

- Enable or disable the specified SPI interrupts.

Parameters:

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- `__INTERRUPT__`: specifies the interrupt source to 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:

- Check whether the specified SPI interrupt source is enabled or not.

Parameters:

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- `__INTERRUPT__`: specifies the SPI interrupt source to check. This parameter can be one of the following values:
 - `SPI_IT_TXE`: Tx buffer empty interrupt enable
 - `SPI_IT_RXNE`: RX buffer not empty interrupt enable
 - `SPI_IT_ERR`: Error interrupt enable

Return value:

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

[__HAL_SPI_GET_FLAG](#)**Description:**

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

Parameters:

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `SPI_FLAG_RXNE`: Receive buffer not empty flag
 - `SPI_FLAG_TXE`: Transmit buffer empty flag
 - `SPI_FLAG_CRCERR`: CRC error flag
 - `SPI_FLAG_MODF`: Mode fault flag
 - `SPI_FLAG_OVR`: Overrun flag
 - `SPI_FLAG_BSY`: Busy flag
 - `SPI_FLAG_FRE`: Frame format error flag
 - `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:**

- Clear the SPI CRCERR pending flag.

Parameters:

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

`__HAL_SPI_CLEAR_MODFLAG`

Description:

- Clear the SPI MODF pending flag.

Parameters:

- `__HANDLE__`: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

`__HAL_SPI_CLEAR_OVRFAG`

Description:

- Clear the SPI OVR pending flag.

Parameters:

- `__HANDLE__`: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

`__HAL_SPI_CLEAR_FREFLAG`

Description:

- Clear the SPI FRE pending flag.

Parameters:

- `__HANDLE__`: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

`__HAL_SPI_ENABLE`

Description:

- Enable the SPI peripheral.

Parameters:

- `__HANDLE__`: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

`__HAL_SPI_DISABLE`

Description:

- Disable the SPI peripheral.

Parameters:

- `__HANDLE__`: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

SPI 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 Mode

SPI_NSS_PULSE_ENABLE
SPI_NSS_PULSE_DISABLE
IS_SPI_NSSP

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_DISABLE

SPI_TIMODE_ENABLE

IS_SPI_TIMODE

SPI Transmission FIFO Status Level

SPI_FTLVL_EMPTY

SPI_FTLVL_QUARTER_FULL

SPI_FTLVL_HALF_FULL

SPI_FTLVL_FULL

38 HAL SPI Extension Driver

38.1 SPIEx Firmware driver API description

38.1.1 IO operation functions

This subsection provides a set of extended functions to manage the SPI data transfers.

1. Rx data flush function:
 - `HAL_SPIExFlushRx_fifo()`

This section contains the following APIs:

- `HAL_SPIExFlushRx_fifo()`

38.1.2 `HAL_SPIExFlushRx_fifo`

Function Name	<code>HAL_StatusTypeDef HAL_SPIExFlushRx_fifo(SPI_HandleTypeDef * hspi)</code>
Function Description	Flush the RX fifo.
Parameters	<ul style="list-style-type: none">• <code>hspi</code>: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none">• HAL status

39 HAL TIM Generic Driver

39.1 TIM Firmware driver registers structures

39.1.1 TIM_Base_InitTypeDef

Data Fields

- *uint32_t Prescaler*
- *uint32_t CounterMode*
- *uint32_t Period*
- *uint32_t ClockDivision*
- *uint32_t RepetitionCounter*

Field Documentation

- ***uint32_t TIM_Base_InitTypeDef::Prescaler***
Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_Base_InitTypeDef::CounterMode***
Specifies the counter mode. This parameter can be a value of [**TIM_Counter_Mode**](#)
- ***uint32_t TIM_Base_InitTypeDef::Period***
Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::ClockDivision***
Specifies the clock division. This parameter can be a value of [**TIM_ClockDivision**](#)
- ***uint32_t TIM_Base_InitTypeDef::RepetitionCounter***
Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to the number of PWM periods in edge-aligned mode the number of half PWM period in center-aligned mode. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
Note:This parameter is valid only for TIM1 and TIM8.

39.1.2 TIM_OC_InitTypeDef

Data Fields

- *uint32_t OCMode*
- *uint32_t Pulse*
- *uint32_t OCIdleState*
- *uint32_t OCNPolarity*
- *uint32_t OCNPolarity*
- *uint32_t OCFastMode*
- *uint32_t OCIdleState*
- *uint32_t OCIdleState*

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode***
Specifies the TIM mode. This parameter can be a value of
TIM_Output_Compare_and_PWM_modes
- ***uint32_t TIM_OC_InitTypeDef::Pulse***
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OC_InitTypeDef::OCPolarity***
Specifies the output polarity. This parameter can be a value of
TIM_Output_Compare_Polarity
- ***uint32_t TIM_OC_InitTypeDef::OCNPolarity***
Specifies the complementary output polarity. This parameter can be a value of
TIM_Output_Compare_N_Polarity
Note:This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***
Specifies the Fast mode state. This parameter can be a value of
TIM_Output_Fast_State
Note:This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32_t TIM_OC_InitTypeDef::OCIdleState***
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of
TIM_Output_Compare_Idle_State
Note:This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OC_InitTypeDef::OCNIdleState***
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of
TIM_Output_Compare_N_Idle_State
Note:This parameter is valid only for TIM1 and TIM8.

39.1.3 TIM_OnePulse_InitTypeDef

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

39.1.4 TIM_IC_InitTypeDef

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_IC_InitTypeDef::ICPolarity***
Specifies the active edge of the input signal. This parameter can be a value of [**TIM_Input_Capture_Polarity**](#)
- ***uint32_t TIM_IC_InitTypeDef::ICSelection***
Specifies the input. This parameter can be a value of [**TIM_Input_Capture_Selection**](#)
- ***uint32_t TIM_IC_InitTypeDef::ICPrescaler***
Specifies the Input Capture Prescaler. This parameter can be a value of [**TIM_Input_Capture_Prescaler**](#)
- ***uint32_t TIM_IC_InitTypeDef::ICFilter***
Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

39.1.5 TIM_Encoder_InitTypeDef

Data Fields

- *uint32_t EncoderMode*
- *uint32_t IC1Polarity*
- *uint32_t IC1Selection*
- *uint32_t IC1Prescaler*
- *uint32_t IC1Filter*
- *uint32_t IC2Polarity*
- *uint32_t IC2Selection*
- *uint32_t IC2Prescaler*
- *uint32_t IC2Filter*

Field Documentation

- *uint32_t TIM_Encoder_InitTypeDef::EncoderMode*
Specifies the active edge of the input signal. This parameter can be a value of [**TIM_Encoder_Mode**](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC1Polarity*
Specifies the active edge of the input signal. This parameter can be a value of [**TIM_Input_Capture_Polarity**](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC1Selection*
Specifies the input. This parameter can be a value of [**TIM_Input_Capture_Selection**](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC1Prescaler*
Specifies the Input Capture Prescaler. This parameter can be a value of [**TIM_Input_Capture_Prescaler**](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC1Filter*
Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- *uint32_t TIM_Encoder_InitTypeDef::IC2Polarity*
Specifies the active edge of the input signal. This parameter can be a value of [**TIM_Input_Capture_Polarity**](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC2Selection*
Specifies the input. This parameter can be a value of [**TIM_Input_Capture_Selection**](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC2Prescaler*
Specifies the Input Capture Prescaler. This parameter can be a value of [**TIM_Input_Capture_Prescaler**](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC2Filter*
Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

39.1.6 TIM_ClockConfigTypeDef**Data Fields**

- *uint32_t ClockSource*
- *uint32_t ClockPolarity*
- *uint32_t ClockPrescaler*
- *uint32_t ClockFilter*

Field Documentation

- ***uint32_t TIM_ClockConfigTypeDef::ClockSource***
TIM clock sources This parameter can be a value of [**TIM_Clock_Source**](#)
- ***uint32_t TIM_ClockConfigTypeDef::ClockPolarity***
TIM clock polarity This parameter can be a value of [**TIM_Clock_Polarity**](#)
- ***uint32_t TIM_ClockConfigTypeDef::ClockPrescaler***
TIM clock prescaler This parameter can be a value of [**TIM_Clock_Prescaler**](#)
- ***uint32_t TIM_ClockConfigTypeDef::ClockFilter***
TIM clock filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

39.1.7 TIM_ClearInputConfigTypeDef

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***
TIM clear Input state This parameter can be ENABLE or DISABLE
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***
TIM clear Input sources This parameter can be a value of [**TIM_ClearInput_Source**](#)
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***
TIM Clear Input polarity This parameter can be a value of [**TIM_ClearInput_Polarity**](#)
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***
TIM Clear Input prescaler This parameter can be a value of [**TIM_ClearInput_Prescaler**](#)
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***
TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

39.1.8 TIM_SlaveConfigTypeDef

Data Fields

- ***uint32_t SlaveMode***
- ***uint32_t InputTrigger***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerPrescaler***
- ***uint32_t TriggerFilter***

Field Documentation

- ***uint32_t TIM_SlaveConfigTypeDef::SlaveMode***
Slave mode selection This parameter can be a value of [**TIM_Slave_Mode**](#)

- ***uint32_t TIM_SlaveConfigTypeDef::InputTrigger***
Input Trigger source This parameter can be a value of [**TIM_Trigger_Selection**](#)
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity***
Input Trigger polarity This parameter can be a value of [**TIM_Trigger_Polarity**](#)
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler***
Input trigger prescaler This parameter can be a value of [**TIM_Trigger_Prescaler**](#)
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerFilter***
Input trigger filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

39.1.9 TIM_HandleTypeDef

Data Fields

- ***TIM_TypeDef * Instance***
- ***TIM_Base_InitTypeDef Init***
- ***HAL_TIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_TIM_StateTypeDef State***

Field Documentation

- ***TIM_TypeDef* TIM_HandleTypeDef::Instance***
Register base address
- ***TIM_Base_InitTypeDef TIM_HandleTypeDef::Init***
TIM Time Base required parameters
- ***HAL_TIM_ActiveChannel TIM_HandleTypeDef::Channel***
Active channel
- ***DMA_HandleTypeDef* TIM_HandleTypeDef::hdma[7]***
DMA Handlers array This array is accessed by a [**TIM_DMA_Handle_index**](#)
- ***HAL_LockTypeDef TIM_HandleTypeDef::Lock***
Locking object
- ***__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State***
TIM operation state

39.2 TIM Firmware driver API description

39.2.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
 - Input Capture
 - Output Compare
 - PWM generation (Edge and Center-aligned Mode)
 - One-pulse mode output

39.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
 - Time Base : HAL_TIM_Base_MspInit()
 - Input Capture : HAL_TIM_IC_MspInit()
 - Output Compare : HAL_TIM_OC_MspInit()
 - PWM generation : HAL_TIM_PWM_MspInit()
 - One-pulse mode output : HAL_TIM_OnePulse_MspInit()
 - Encoder mode output : HAL_TIM_Encoder_MspInit()
2. Initialize the TIM low level resources :
 - a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
 - b. TIM pins configuration
 - Enable the clock for the TIM GPIOs using the following function:
__HAL_RCC_GPIOx_CLK_ENABLE();
 - Configure these TIM pins in Alternate function mode using
HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
 - HAL_TIM_Base_Init: to use the Timer to generate a simple time base
 - HAL_TIM_OC_Init and HAL_TIM_OC_ConfigChannel: to use the Timer to generate an Output Compare signal.
 - HAL_TIM_PWM_Init and HAL_TIM_PWM_ConfigChannel: to use the Timer to generate a PWM signal.
 - HAL_TIM_IC_Init and HAL_TIM_IC_ConfigChannel: to use the Timer to measure an external signal.
 - HAL_TIM_OnePulse_Init and HAL_TIM_OnePulse_ConfigChannel: to use the Timer in One Pulse Mode.
 - HAL_TIM_Encoder_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
 - Time Base : HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(),
HAL_TIM_Base_Start_IT()
 - Input Capture : HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(),
HAL_TIM_IC_Start_IT()
 - Output Compare : HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(),
HAL_TIM_OC_Start_IT()
 - PWM generation : HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(),
HAL_TIM_PWM_Start_IT()
 - One-pulse mode output : HAL_TIM_OnePulse_Start(),
HAL_TIM_OnePulse_Start_IT()
 - Encoder mode output : HAL_TIM_Encoder_Start(),
HAL_TIM_Encoder_Start_DMA(), HAL_TIM_Encoder_Start_IT().
6. The DMA Burst is managed with the two following functions:
HAL_TIM_DMABurst_WriteStart() HAL_TIM_DMABurst_ReadStart()

39.2.3 Time Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.

- De-initialize the TIM base.
- Start the Time Base.
- Stop the Time Base.
- Start the Time Base and enable interrupt.
- Stop the Time Base and disable interrupt.
- Start the Time Base and enable DMA transfer.
- Stop the Time Base and disable DMA transfer.

This section contains the following APIs:

- [*HAL_TIM_Base_Init\(\)*](#)
- [*HAL_TIM_Base_DeInit\(\)*](#)
- [*HAL_TIM_Base_MspInit\(\)*](#)
- [*HAL_TIM_Base_MspDeInit\(\)*](#)
- [*HAL_TIM_Base_Start\(\)*](#)
- [*HAL_TIM_Base_Stop\(\)*](#)
- [*HAL_TIM_Base_Start_IT\(\)*](#)
- [*HAL_TIM_Base_Stop_IT\(\)*](#)
- [*HAL_TIM_Base_Start_DMA\(\)*](#)
- [*HAL_TIM_Base_Stop_DMA\(\)*](#)

39.2.4 Time Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the Time Output Compare.
- Stop the Time Output Compare.
- Start the Time Output Compare and enable interrupt.
- Stop the Time Output Compare and disable interrupt.
- Start the Time Output Compare and enable DMA transfer.
- Stop the Time Output Compare and disable DMA transfer.

This section contains the following APIs:

- [*HAL_TIM_OC_Init\(\)*](#)
- [*HAL_TIM_OC_DeInit\(\)*](#)
- [*HAL_TIM_OC_MspInit\(\)*](#)
- [*HAL_TIM_OC_MspDeInit\(\)*](#)
- [*HAL_TIM_OC_Start\(\)*](#)
- [*HAL_TIM_OC_Stop\(\)*](#)
- [*HAL_TIM_OC_Start_IT\(\)*](#)
- [*HAL_TIM_OC_Stop_IT\(\)*](#)
- [*HAL_TIM_OC_Start_DMA\(\)*](#)
- [*HAL_TIM_OC_Stop_DMA\(\)*](#)

39.2.5 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.

This section contains the following APIs:

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

39.2.6 Time Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the Time Input Capture.
- Stop the Time Input Capture.
- Start the Time Input Capture and enable interrupt.
- Stop the Time Input Capture and disable interrupt.
- Start the Time Input Capture and enable DMA transfer.
- Stop the Time Input Capture and disable DMA transfer.

This section contains the following APIs:

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

39.2.7 Time One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the Time One Pulse.
- Stop the Time One Pulse.
- Start the Time One Pulse and enable interrupt.
- Stop the Time One Pulse and disable interrupt.
- Start the Time One Pulse and enable DMA transfer.
- Stop the Time One Pulse and disable DMA transfer.

This section contains the following APIs:

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

39.2.8 Time Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the Time Encoder.
- Stop the Time Encoder.
- Start the Time Encoder and enable interrupt.
- Stop the Time Encoder and disable interrupt.
- Start the Time Encoder and enable DMA transfer.
- Stop the Time Encoder and disable DMA transfer.

This section contains the following APIs:

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

39.2.9 IRQ handler management

This section provides Timer IRQ handler function.

This section contains the following APIs:

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

39.2.10 Peripheral Control functions

This section provides functions allowing to:

- Configure The Input Output channels for OC, PWM, IC or One Pulse mode.
- Configure External Clock source.
- Configure Complementary channels, break features and dead time.
- Configure Master and the Slave synchronization.
- Configure the DMA Burst Mode.

This section contains the following APIs:

- [*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_ConfigT1Input\(\)*](#)
- [*HAL_TIM_SlaveConfigSynchronization\(\)*](#)
- [*HAL_TIM_SlaveConfigSynchronization_IT\(\)*](#)
- [*HAL_TIM_ReadCapturedValue\(\)*](#)

39.2.11 TIM Callbacks functions

This section provides TIM callback functions:

- Timer Period elapsed callback
- Timer Output Compare callback
- Timer Input capture callback
- Timer Trigger callback
- Timer Error callback

This section contains the following APIs:

- [*HAL_TIM_PeriodElapsedCallback\(\)*](#)
- [*HAL_TIM_OC_DelayElapsedCallback\(\)*](#)
- [*HAL_TIM_IC_CaptureCallback\(\)*](#)
- [*HAL_TIM_PWM_PulseFinishedCallback\(\)*](#)
- [*HAL_TIM_TriggerCallback\(\)*](#)
- [*HAL_TIM_ErrorCallback\(\)*](#)

39.2.12 Peripheral State functions

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

This section contains the following APIs:

- [*HAL_TIM_Base_GetState\(\)*](#)
- [*HAL_TIM_OC_GetState\(\)*](#)
- [*HAL_TIM_PWM_GetState\(\)*](#)
- [*HAL_TIM_IC_GetState\(\)*](#)
- [*HAL_TIM_OnePulse_GetState\(\)*](#)
- [*HAL_TIM_Encoder_GetState\(\)*](#)

39.2.13 HAL_TIM_Base_Init

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Base handle
Return values	<ul style="list-style-type: none"> • HAL status

39.2.14 HAL_TIM_Base_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM Base peripheral.
Parameters	<ul style="list-style-type: none">• htim: : TIM Base handle
Return values	<ul style="list-style-type: none">• HAL status

39.2.15 HAL_TIM_Base_MspInit

Function Name	void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Base MSP.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle
Return values	<ul style="list-style-type: none">• None

39.2.16 HAL_TIM_Base_MspDeInit

Function Name	void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Base MSP.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle
Return values	<ul style="list-style-type: none">• None

39.2.17 HAL_TIM_Base_Start

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)
Function Description	Starts the TIM Base generation.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle
Return values	<ul style="list-style-type: none">• HAL status

39.2.18 HAL_TIM_Base_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)
Function Description	Stops the TIM Base generation.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle
Return values	<ul style="list-style-type: none">• HAL status

39.2.19 HAL_TIM_Base_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)
Function Description	Starts the TIM Base generation in interrupt mode.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle
Return values	<ul style="list-style-type: none">• HAL status

39.2.20 HAL_TIM_Base_Stop_IT

Function Name **HAL_StatusTypeDef HAL_TIM_Base_Stop_IT
(TIM_HandleTypeDef * htim)**

Function Description Stops the TIM Base generation in interrupt mode.

Parameters • **htim:** : TIM handle

Return values • HAL status

39.2.21 HAL_TIM_Base_Start_DMA

Function Name **HAL_StatusTypeDef HAL_TIM_Base_Start_DMA
(TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)**

Function Description Starts the TIM Base generation in DMA mode.

Parameters • **htim:** : TIM handle
• **pData:** : The source Buffer address.
• **Length:** : The length of data to be transferred from memory to peripheral.

Return values • HAL status

39.2.22 HAL_TIM_Base_Stop_DMA

Function Name **HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA
(TIM_HandleTypeDef * htim)**

Function Description Stops the TIM Base generation in DMA mode.

Parameters • **htim:** : TIM handle

Return values • HAL status

39.2.23 HAL_TIM_OC_Init

Function Name **HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)**

Function Description Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.

Parameters • **htim:** : TIM Output Compare handle

Return values • HAL status

39.2.24 HAL_TIM_OC_DelInit

Function Name **HAL_StatusTypeDef HAL_TIM_OC_DelInit
(TIM_HandleTypeDef * htim)**

Function Description DelInitializes the TIM peripheral.

Parameters • **htim:** : TIM Output Compare handle

Return values • HAL status

39.2.25 HAL_TIM_OC_MspInit

Function Name	void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

39.2.26 HAL_TIM_OC_MspDeInit

Function Name	void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	Deinitializes TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

39.2.27 HAL_TIM_OC_Start

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Output Compare handle • Channel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.28 HAL_TIM_OC_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.29 HAL_TIM_OC_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Output Compare signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM OC handle

- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

39.2.30 HAL_TIM_OC_Stop_IT

Function Name

**HAL_StatusTypeDef HAL_TIM_OC_Stop_IT
(TIM_HandleTypeDef * htim, uint32_t Channel)**

Function Description

Stops the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

39.2.31 HAL_TIM_OC_Start_DMA

Function Name

**HAL_StatusTypeDef HAL_TIM_OC_Start_DMA
(TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t *
pData, uint16_t Length)**

Function Description

Starts the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** : The source Buffer address.
- **Length:** : The length of data to be transferred from memory to TIM peripheral

Return values

- HAL status

39.2.32 HAL_TIM_OC_Stop_DMA

Function Name

**HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA
(TIM_HandleTypeDef * htim, uint32_t Channel)**

Function Description

Stops the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

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

39.2.33 HAL_TIM_PWM_Init

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • HAL status

39.2.34 HAL_TIM_PWM_DelInit

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_DelInit (TIM_HandleTypeDef * htim)
Function Description	Deinitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • HAL status

39.2.35 HAL_TIM_PWM_MspInit

Function Name	void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM PWM MSP.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

39.2.36 HAL_TIM_PWM_MspDelInit

Function Name	void HAL_TIM_PWM_MspDelInit (TIM_HandleTypeDef * htim)
Function Description	Deinitializes TIM PWM MSP.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

39.2.37 HAL_TIM_PWM_Start

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.38 HAL_TIM_PWM_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.39 HAL_TIM_PWM_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.40 HAL_TIM_PWM_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.41 HAL_TIM_PWM_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2

	selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected <ul style="list-style-type: none"> • pData: : The source Buffer address. • Length: : The length of data to be transferred from memory to TIM peripheral
Return values	<ul style="list-style-type: none"> • HAL status

39.2.42 HAL_TIM_PWM_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.43 HAL_TIM_IC_Init

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Input Capture handle
Return values	<ul style="list-style-type: none"> • HAL status

39.2.44 HAL_TIM_IC_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)
Function Description	Deinitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Input Capture handle
Return values	<ul style="list-style-type: none"> • HAL status

39.2.45 HAL_TIM_IC_MspInit

Function Name	void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

39.2.46 HAL_TIM_IC_MspDeInit

Function Name	void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

39.2.47 HAL_TIM_IC_Start

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Input Capture handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.48 HAL_TIM_IC_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.49 HAL_TIM_IC_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Input Capture handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.50 HAL_TIM_IC_Stop_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Stops the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.51 HAL_TIM_IC_Start_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</code>
Function Description	Starts the TIM Input Capture measurement in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Input Capture handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected • pData: : The destination Buffer address. • Length: : The length of data to be transferred from TIM peripheral to memory.
Return values	<ul style="list-style-type: none"> • HAL status

39.2.52 HAL_TIM_IC_Stop_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Stops the TIM Input Capture measurement in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Input Capture handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.53 HAL_TIM_OnePulse_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)</code>
Function Description	Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated

handle.

Parameters	<ul style="list-style-type: none"> htim: : TIM OnePulse handle OnePulseMode: : Select the One pulse mode. This parameter can be one of the following values: TIM_OPMODE_SINGLE: Only one pulse will be generated.TIM_OPMODE_REPETITIVE: Repetitive pulses will be generated.
Return values	<ul style="list-style-type: none"> HAL status

39.2.54 HAL_TIM_OnePulse_DelInit

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_DelInit (TIM_HandleTypeDef * htim)
Function Description	Deinitializes the TIM One Pulse.
Parameters	<ul style="list-style-type: none"> htim: : TIM One Pulse handle
Return values	<ul style="list-style-type: none"> HAL status

39.2.55 HAL_TIM_OnePulse_MspInit

Function Name	void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"> htim: : TIM handle
Return values	<ul style="list-style-type: none"> None

39.2.56 HAL_TIM_OnePulse_MspDelInit

Function Name	void HAL_TIM_OnePulse_MspDelInit (TIM_HandleTypeDef * htim)
Function Description	Deinitializes TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"> htim: : TIM handle
Return values	<ul style="list-style-type: none"> None

39.2.57 HAL_TIM_OnePulse_Start

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Starts the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> htim: : TIM One Pulse handle OutputChannel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> HAL status

39.2.58 HAL_TIM_OnePulse_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Stops the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> • htim: : TIM One Pulse handle • OutputChannel: : TIM Channels to be disable This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.59 HAL_TIM_OnePulse_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Starts the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM One Pulse handle • OutputChannel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.60 HAL_TIM_OnePulse_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Stops the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM One Pulse handle • OutputChannel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.61 HAL_TIM_Encoder_Init

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)
Function Description	Initializes the TIM Encoder Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • sConfig: : TIM Encoder Interface configuration structure
Return values	<ul style="list-style-type: none"> • HAL status

39.2.62 HAL_TIM_Encoder_DelInit

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_DelInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM Encoder interface.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder handle
Return values	<ul style="list-style-type: none"> • HAL status

39.2.63 HAL_TIM_Encoder_MspInit

Function Name	void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

39.2.64 HAL_TIM_Encoder_MspDelInit

Function Name	void HAL_TIM_Encoder_MspDelInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

39.2.65 HAL_TIM_Encoder_Start

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Encoder Interface.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.66 HAL_TIM_Encoder_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Encoder Interface.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values	<ul style="list-style-type: none"> • HAL status
39.2.67 HAL_TIM_Encoder_Start_IT	
Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Encoder Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	<ul style="list-style-type: none"> • HAL status
39.2.68 HAL_TIM_Encoder_Stop_IT	
Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Encoder Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	<ul style="list-style-type: none"> • HAL status
39.2.69 HAL_TIM_Encoder_Start_DMA	
Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)
Function Description	Starts the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected • pData1: : The destination Buffer address for IC1. • pData2: : The destination Buffer address for IC2. • Length: : The length of data to be transferred from TIM peripheral to memory.
Return values	<ul style="list-style-type: none"> • HAL status

39.2.70 HAL_TIM_Encoder_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.71 HAL_TIM_IRQHandler

Function Name	void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)
Function Description	This function handles TIM interrupts requests.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

39.2.72 HAL_TIM_OC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)
Function Description	Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Output Compare handle • sConfig: : TIM Output Compare configuration structure • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

39.2.73 HAL_TIM_IC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig, uint32_t Channel)
Function Description	Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.
Parameters	<ul style="list-style-type: none"> • htim: : TIM IC handle • sConfig: : TIM Input Capture configuration structure • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3

selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values • HAL status

39.2.74 HAL_TIM_PWM_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)
Function Description	Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • sConfig: : TIM PWM configuration structure • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.75 HAL_TIM_OnePulse_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)
Function Description	Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.
Parameters	<ul style="list-style-type: none"> • htim: : TIM One Pulse handle • sConfig: : TIM One Pulse configuration structure • OutputChannel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_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 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	• HAL status

39.2.76 HAL_DMABurst_WriteStart

Function Name	HAL_StatusTypeDef HAL_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)
Function Description	Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • BurstBaseAddress: : TIM Base address from where the DMA will start the Data write This parameter can be one of the following values: TIM_DMABASE_CR1TIM_DMABASE_CR2TIM_DMABASE_SMCRTIM_DMABASE_DIERTIM_DMABASE_SRTIM_DMABASE_EGRTIM_D

MABASE_CCMR1TIM_DMABASE_CCMR2TIM_DMABASE_CCERTI
 M_DMABASE_CNTTIM_DMABASE_PSCTIM_DMABASE_ARRTIM_
 DMABASE_RCRTIM_DMABASE_CCR1TIM_DMABASE_CCR2TIM_
 DMABASE_CCR3TIM_DMABASE_CCR4TIM_DMABASE_BDRTIM_
 DMABASE_DCR

- **BurstRequestSrc:** : TIM DMA Request sources This parameter can be one of the following values: TIM_DMA_UPDATE: TIM update Interrupt sourceTIM_DMA_CC1: TIM Capture Compare 1 DMA sourceTIM_DMA_CC2: TIM Capture Compare 2 DMA sourceTIM_DMA_CC3: TIM Capture Compare 3 DMA sourceTIM_DMA_CC4: TIM Capture Compare 4 DMA sourceTIM_DMA_COM: TIM Commutation DMA sourceTIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** : The Buffer address.
- **BurstLength:** : DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.

Return values • HAL status

39.2.77 HAL_TIM_DMABurst_WriteStop

Function Name **HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop
 (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)**

Function Description Stops the TIM DMA Burst mode.

Parameters • **htim:** : TIM handle
 • **BurstRequestSrc:** : TIM DMA Request sources to disable

Return values • HAL status

39.2.78 HAL_TIM_DMABurst_ReadStart

Function Name **HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart
 (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t
 BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)**

Function Description Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters • **htim:** : TIM handle
 • **BurstBaseAddress:** : TIM Base address from where the DMA will starts the Data read This parameter can be one of the following values:
 TIM_DMABASE_CR1TIM_DMABASE_CR2TIM_DMABASE_SMCR
 TIM_DMABASE_DIERTIM_DMABASE_SRTIM_DMABASE_EGRTI
 M_DMABASE_CCMR1TIM_DMABASE_CCMR2TIM_DMABASE_C
 CERTIM_DMABASE_CNTTIM_DMABASE_PSCTIM_DMABASE_A
 RRTIM_DMABASE_RCRTIM_DMABASE_CCR1TIM_DMABASE_C
 CR2TIM_DMABASE_CCR3TIM_DMABASE_CCR4TIM_DMABASE_BDRTIM_DMABASE_DCR
 • **BurstRequestSrc:** : TIM DMA Request sources This parameter can be one of the following values: TIM_DMA_UPDATE: TIM update Interrupt sourceTIM_DMA_CC1: TIM Capture Compare 1 DMA sourceTIM_DMA_CC2: TIM Capture Compare 2 DMA sourceTIM_DMA_CC3: TIM Capture Compare 3 DMA

- sourceTIM_DMA_CC4: TIM Capture Compare 4 DMA
 - sourceTIM_DMA_COM: TIM Commutation DMA
 - sourceTIM_DMA_TRIGGER: TIM Trigger DMA source
 - **BurstBuffer:** : The Buffer address.
 - **BurstLength:** : DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.
- Return values**
- HAL status

39.2.79 HAL_TIM_DMABurst_ReadStop

Function Name	HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)
Function Description	Stop the DMA burst reading.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • BurstRequestSrc: : TIM DMA Request sources to disable.
Return values	<ul style="list-style-type: none"> • HAL status

39.2.80 HAL_TIM_GenerateEvent

Function Name	HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)
Function Description	Generate a software event.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • EventSource: : specifies the event source. This parameter can be one of the following values: TIM_EVENTSOURCE_UPDATE: Timer update Event sourceTIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event sourceTIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event sourceTIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event sourceTIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event sourceTIM_EVENTSOURCE_COM: Timer COM event sourceTIM_EVENTSOURCE_TRIGGER: Timer Trigger Event sourceTIM_EVENTSOURCE_BREAK: Timer Break event source
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • TIM6 and TIM7 can only generate an update event. • TIM_EVENTSOURCE_COM and TIM_EVENTSOURCE_BREAK are used only with TIM1, TIM15, TIM16 and TIM17.

39.2.81 HAL_TIM_ConfigOCrefClear

Function Name	HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)
Function Description	Configures the OCRef clear feature.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle

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

Return values

- HAL status

39.2.82 HAL_TIM_ConfigClockSource

Function Name	HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)
Function Description	Configures the clock source to be used.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • sClockSourceConfig: : pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.
Return values	<ul style="list-style-type: none"> • HAL status

39.2.83 HAL_TIM_ConfigTI1Input

Function Name	HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)
Function Description	Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle. • TI1_Selection: : Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values: TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 inputTIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)
Return values	<ul style="list-style-type: none"> • HAL status

39.2.84 HAL_TIM_SlaveConfigSynchronization

Function Name	HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)
Function Description	Configures the TIM in Slave mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle. • sSlaveConfig: : pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External

clock mode 1).

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

39.2.85 HAL_TIM_SlaveConfigSynchronization_IT

Function Name	HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)
Function Description	Configures the TIM in Slave mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle. • sSlaveConfig: pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	<ul style="list-style-type: none"> • HAL status

39.2.86 HAL_TIM_ReadCapturedValue

Function Name	uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Read the captured value from Capture Compare unit.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle. • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1 : TIM Channel 1 selectedTIM_CHANNEL_2 : TIM Channel 2 selectedTIM_CHANNEL_3 : TIM Channel 3 selectedTIM_CHANNEL_4 : TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • Captured value

39.2.87 HAL_TIM_PeriodElapsedCallback

Function Name	void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)
Function Description	Period elapsed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

39.2.88 HAL_TIM_OC_DelayElapsedCallback

Function Name	void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)
Function Description	Output Compare callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM OC handle
Return values	<ul style="list-style-type: none"> • None

39.2.89 HAL_TIM_IC_CaptureCallback

Function Name	void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)
Function Description	Input Capture callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• htim: : TIM IC handle
Return values	<ul style="list-style-type: none">• None

39.2.90 HAL_TIM_PWM_PulseFinishedCallback

Function Name	void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)
Function Description	PWM Pulse finished callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle
Return values	<ul style="list-style-type: none">• None

39.2.91 HAL_TIM_TriggerCallback

Function Name	void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)
Function Description	Hall Trigger detection callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle
Return values	<ul style="list-style-type: none">• None

39.2.92 HAL_TIM_ErrorCallback

Function Name	void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)
Function Description	Timer error callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle
Return values	<ul style="list-style-type: none">• None

39.2.93 HAL_TIM_Base_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Base state.
Parameters	<ul style="list-style-type: none">• htim: : TIM Base handle
Return values	<ul style="list-style-type: none">• HAL state

39.2.94 HAL_TIM_OC_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM OC state.
Parameters	<ul style="list-style-type: none">• htim: : TIM Ouput Compare handle
Return values	<ul style="list-style-type: none">• HAL state

39.2.95 HAL_TIM_PWM_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM PWM state.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • HAL state

39.2.96 HAL_TIM_IC_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Input Capture state.
Parameters	<ul style="list-style-type: none"> • htim: : TIM IC handle
Return values	<ul style="list-style-type: none"> • HAL state

39.2.97 HAL_TIM_OnePulse_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM One Pulse Mode state.
Parameters	<ul style="list-style-type: none"> • htim: : TIM OPM handle
Return values	<ul style="list-style-type: none"> • HAL state

39.2.98 HAL_TIM_Encoder_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Encoder Mode state.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder handle
Return values	<ul style="list-style-type: none"> • HAL state

39.3 TIM Firmware driver defines**39.3.1 TIM*****TIM Automatic Output Enable***

TIM_AUTOMATICOUTPUT_ENABLE

TIM_AUTOMATICOUTPUT_DISABLE

TIM Break Input Enable Disable

TIM_BREAK_ENABLE

TIM_BREAK_DISABLE

TIM Break Input Polarity

TIM_BREAKPOLARITY_LOW

TIM_BREAKPOLARITY_HIGH

TIM Channel

TIM_CHANNEL_1

TIM_CHANNEL_2

TIM_CHANNEL_3

TIM_CHANNEL_4

TIM_CHANNEL_ALL

TIM Capture/Compare Channel State

TIM_CCx_ENABLE

TIM_CCx_DISABLE

TIM_CCxN_ENABLE

TIM_CCxN_DISABLE

TIM Clear Input Polarity

TIM_CLEARINPUTPOLARITY_INVERTED Polarity for ETRx pin

TIM_CLEARINPUTPOLARITY_NONINVERTED Polarity for ETRx pin

TIM Clear Input Prescaler

TIM_CLEARINPUTPRESCALER_DIV1 No prescaler is used

TIM_CLEARINPUTPRESCALER_DIV2 Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_CLEARINPUTPRESCALER_DIV4 Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_CLEARINPUTPRESCALER_DIV8 Prescaler for External ETR pin: Capture performed once every 8 events.

TIM ClearInput Source

TIM_CLEARINPUTSOURCE_ETR

TIM_CLEARINPUTSOURCE_NONE

TIM Clock Division

TIM_CLOCKDIVISION_DIV1

TIM_CLOCKDIVISION_DIV2

TIM_CLOCKDIVISION_DIV4

TIM Clock Polarity

TIM_CLOCKPOLARITY_INVERTED Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_NONINVERTED Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_RISING Polarity for TIx clock sources

TIM_CLOCKPOLARITY_FALLING Polarity for TIx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE Polarity for TIx clock sources

TIM Clock Prescaler

TIM_CLOCKPRESCALER_DIV1	No prescaler is used
TIM_CLOCKPRESCALER_DIV2	Prescaler for External ETR Clock: Capture performed once every 2 events.
TIM_CLOCKPRESCALER_DIV4	Prescaler for External ETR Clock: Capture performed once every 4 events.
TIM_CLOCKPRESCALER_DIV8	Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_ETRMODE2
TIM_CLOCKSOURCE_INTERNAL
TIM_CLOCKSOURCE_ITR0
TIM_CLOCKSOURCE_ITR1
TIM_CLOCKSOURCE_ITR2
TIM_CLOCKSOURCE_ITR3
TIM_CLOCKSOURCE_TI1ED
TIM_CLOCKSOURCE_TI1
TIM_CLOCKSOURCE_TI2
TIM_CLOCKSOURCE_ETRMODE1

TIM Commutation Source

TIM_COMMUTATION_TRGI
TIM_COMMUTATION_SOFTWARE

TIM Counter Mode

TIM_COUNTERMODE_UP
TIM_COUNTERMODE_DOWN
TIM_COUNTERMODE_CENTERALIGNED1
TIM_COUNTERMODE_CENTERALIGNED2
TIM_COUNTERMODE_CENTERALIGNED3

TIM DMA Base Address

TIM_DMABASE_CR1
TIM_DMABASE_CR2
TIM_DMABASE_SMCR
TIM_DMABASE_DIER
TIM_DMABASE_SR
TIM_DMABASE_EGR
TIM_DMABASE_CCMR1
TIM_DMABASE_CCMR2
TIM_DMABASE_CCER

TIM_DMABASE_CNT
TIM_DMABASE_PSC
TIM_DMABASE_ARR
TIM_DMABASE_RCR
TIM_DMABASE_CCR1
TIM_DMABASE_CCR2
TIM_DMABASE_CCR3
TIM_DMABASE_CCR4
TIM_DMABASE_BDTR
TIM_DMABASE_DCR
TIM_DMABASE_OR

TIM DMA Burst Length

TIM_DMABURSTLENGTH_1TRANSFER
TIM_DMABURSTLENGTH_2TRANSFERS
TIM_DMABURSTLENGTH_3TRANSFERS
TIM_DMABURSTLENGTH_4TRANSFERS
TIM_DMABURSTLENGTH_5TRANSFERS
TIM_DMABURSTLENGTH_6TRANSFERS
TIM_DMABURSTLENGTH_7TRANSFERS
TIM_DMABURSTLENGTH_8TRANSFERS
TIM_DMABURSTLENGTH_9TRANSFERS
TIM_DMABURSTLENGTH_10TRANSFERS
TIM_DMABURSTLENGTH_11TRANSFERS
TIM_DMABURSTLENGTH_12TRANSFERS
TIM_DMABURSTLENGTH_13TRANSFERS
TIM_DMABURSTLENGTH_14TRANSFERS
TIM_DMABURSTLENGTH_15TRANSFERS
TIM_DMABURSTLENGTH_16TRANSFERS
TIM_DMABURSTLENGTH_17TRANSFERS
TIM_DMABURSTLENGTH_18TRANSFERS

TIM DMA Handle Index

TIM_DMA_ID_UPDATE	Index of the DMA handle used for Update DMA requests
TIM_DMA_ID_CC1	Index of the DMA handle used for Capture/Compare 1 DMA 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	
TIM Encoder Mode	
TIM_ENCODERMODE_TI1	
TIM_ENCODERMODE_TI2	
TIM_ENCODERMODE_TI12	
TIM ETR Polarity	
TIM_ETRPOLARITY_INVERTED	Polarity for ETR source
TIM_ETRPOLARITY_NONINVERTED	Polarity for ETR source
TIM ETR Prescaler	
TIM_ETRPRESCALER_DIV1	No prescaler is used
TIM_ETRPRESCALER_DIV2	ETR input source is divided by 2
TIM_ETRPRESCALER_DIV4	ETR input source is divided by 4
TIM_ETRPRESCALER_DIV8	ETR input source is divided by 8
TIM Event Source	
TIM_EVENTSOURCE_UPDATE	
TIM_EVENTSOURCE_CC1	
TIM_EVENTSOURCE_CC2	
TIM_EVENTSOURCE_CC3	
TIM_EVENTSOURCE_CC4	
TIM_EVENTSOURCE_COM	
TIM_EVENTSOURCE_TRIGGER	
TIM_EVENTSOURCE_BREAK	
TIM Exported Macros	
<u>_HAL_TIM_RESET_HANDLE_ST Description:</u>	

ATE

- Reset TIM handle state.

Parameters:

- `_HANDLE_`: TIM handle.

Return value:

- None

`_HAL_TIM_ENABLE`**Description:**

- Enable the TIM peripheral.

Parameters:

- `_HANDLE_`: TIM handle

Return value:

- None

`_HAL_TIM_MOE_ENABLE`**Description:**

- Enable the TIM main Output.

Parameters:

- `_HANDLE_`: TIM handle

Return value:

- None

`_HAL_TIM_DISABLE`**Description:**

- Disable the TIM peripheral.

Parameters:

- `_HANDLE_`: TIM handle

Return value:

- None

`_HAL_TIM_MOE_DISABLE`**Description:**

- Disable the TIM main Output.

Parameters:

- `_HANDLE_`: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled only if all the CCx and CCxN channels have been disabled

`_HAL_TIM_ENABLE_IT`**Description:**

- Enables the specified TIM interrupt.

Parameters:

- `_HANDLE_`: specifies the TIM Handle.

- `__INTERRUPT__`: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
 - `TIM_IT_UPDATE`: Update interrupt
 - `TIM_IT_CC1`: Capture/Compare 1 interrupt
 - `TIM_IT_CC2`: Capture/Compare 2 interrupt
 - `TIM_IT_CC3`: Capture/Compare 3 interrupt
 - `TIM_IT_CC4`: Capture/Compare 4 interrupt
 - `TIM_IT_COM`: Commutation interrupt
 - `TIM_IT_TRIGGER`: Trigger interrupt
 - `TIM_IT_BREAK`: Break interrupt

Return value:

- None

`__HAL_TIM_DISABLE_IT`**Description:**

- Disables the specified TIM interrupt.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__INTERRUPT__`: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
 - `TIM_IT_UPDATE`: Update interrupt
 - `TIM_IT_CC1`: Capture/Compare 1 interrupt
 - `TIM_IT_CC2`: Capture/Compare 2 interrupt
 - `TIM_IT_CC3`: Capture/Compare 3 interrupt
 - `TIM_IT_CC4`: Capture/Compare 4 interrupt
 - `TIM_IT_COM`: Commutation interrupt
 - `TIM_IT_TRIGGER`: Trigger interrupt
 - `TIM_IT_BREAK`: Break interrupt

Return value:

- None

`__HAL_TIM_ENABLE_DMA`**Description:**

- Enables the specified DMA request.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__DMA__`: specifies the TIM DMA request to enable. This parameter can be one of the following values:
 - `TIM_DMA_UPDATE`: Update DMA request
 - `TIM_DMA_CC1`: Capture/Compare 1 DMA request
 - `TIM_DMA_CC2`: Capture/Compare 2 DMA request
 - `TIM_DMA_CC3`: Capture/Compare 3 DMA request
 - `TIM_DMA_CC4`: Capture/Compare 4 DMA request
 - `TIM_DMA_COM`: Commutation DMA

- request
- TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

`_HAL_TIM_DISABLE_DMA`

Description:

- Disables the specified DMA request.

Parameters:

- `_HANDLE_`: specifies the TIM Handle.
- `_DMA_`: specifies the TIM DMA request to disable. This parameter can be one of the following values:
 - TIM_DMA_UPDATE: Update DMA request
 - TIM_DMA_CC1: Capture/Compare 1 DMA request
 - TIM_DMA_CC2: Capture/Compare 2 DMA request
 - TIM_DMA_CC3: Capture/Compare 3 DMA request
 - TIM_DMA_CC4: Capture/Compare 4 DMA request
 - TIM_DMA_COM: Commutation DMA request
 - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

`_HAL_TIM_GET_FLAG`

Description:

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

Parameters:

- `_HANDLE_`: specifies the TIM Handle.
- `_FLAG_`: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
 - TIM_FLAG_UPDATE: Update interrupt flag
 - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
 - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
 - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
 - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
 - TIM_FLAG_COM: Commutation interrupt flag
 - TIM_FLAG_TRIGGER: Trigger interrupt flag

- `TIM_FLAG_BREAK`: Break interrupt flag
- `TIM_FLAG_CC1OF`: Capture/Compare 1 overcapture flag
- `TIM_FLAG_CC2OF`: Capture/Compare 2 overcapture flag
- `TIM_FLAG_CC3OF`: Capture/Compare 3 overcapture flag
- `TIM_FLAG_CC4OF`: Capture/Compare 4 overcapture flag

Return value:

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

`_HAL_TIM_CLEAR_FLAG`**Description:**

- Clears the specified TIM interrupt flag.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__FLAG__`: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
 - `TIM_FLAG_UPDATE`: Update interrupt flag
 - `TIM_FLAG_CC1`: Capture/Compare 1 interrupt flag
 - `TIM_FLAG_CC2`: Capture/Compare 2 interrupt flag
 - `TIM_FLAG_CC3`: Capture/Compare 3 interrupt flag
 - `TIM_FLAG_CC4`: Capture/Compare 4 interrupt flag
 - `TIM_FLAG_COM`: Commutation interrupt flag
 - `TIM_FLAG_TRIGGER`: Trigger interrupt flag
 - `TIM_FLAG_BREAK`: Break interrupt flag
 - `TIM_FLAG_CC1OF`: Capture/Compare 1 overcapture flag
 - `TIM_FLAG_CC2OF`: Capture/Compare 2 overcapture flag
 - `TIM_FLAG_CC3OF`: Capture/Compare 3 overcapture flag
 - `TIM_FLAG_CC4OF`: Capture/Compare 4 overcapture flag

Return value:

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

`_HAL_TIM_GET_IT_SOURCE`**Description:**

- Checks whether the specified TIM interrupt has occurred or not.

Parameters:

- `__HANDLE__`: TIM handle
- `__INTERRUPT__`: specifies the TIM interrupt

source to check.

Return value:

- The state of TIM_IT (SET or RESET).

`_HAL_TIM_CLEAR_IT`

Description:

- Clear the TIM interrupt pending bits.

Parameters:

- `_HANDLE_`: TIM handle
- `_INTERRUPT_`: specifies the interrupt pending bit to clear.

Return value:

- None

`_HAL_TIM_IS_TIM_COUNTING_DOWN`

Description:

- Indicates whether or not the TIM Counter is used as downcounter.

Parameters:

- `_HANDLE_`: TIM handle.

Return value:

- False: (Counter used as upcounter) or True (Counter used as downcounter)

Notes:

- This macro is particularly usefull to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

`_HAL_TIM_SET_PRESCALER`

Description:

- Sets the TIM active prescaler register value on update event.

Parameters:

- `_HANDLE_`: TIM handle.
- `_PRESC_`: specifies the active prescaler register new value.

Return value:

- None

`_HAL_TIM_SET_COMPARE`

Description:

- Sets the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

Parameters:

- `_HANDLE_`: TIM handle.
- `_CHANNEL_`: : TIM Channels to be configured. This parameter can be one of the following values:

- `TIM_CHANNEL_1`: TIM Channel 1 selected
- `TIM_CHANNEL_2`: TIM Channel 2 selected
- `TIM_CHANNEL_3`: TIM Channel 3 selected
- `TIM_CHANNEL_4`: TIM Channel 4 selected
- `_COMPARE_`: specifies the Capture Compare register new value.

Return value:

- None

[__HAL_TIM_GET_COMPARE](#)**Description:**

- Gets the TIM Capture Compare Register value on runtime.

Parameters:

- `_HANDLE_`: TIM handle.
- `_CHANNEL_`: : TIM Channel associated with the capture compare register This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: get capture/compare 1 register value
 - `TIM_CHANNEL_2`: get capture/compare 2 register value
 - `TIM_CHANNEL_3`: get capture/compare 3 register value
 - `TIM_CHANNEL_4`: get capture/compare 4 register value

Return value:

- None

[__HAL_TIM_SET_COUNTER](#)**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_GET_COUNTER](#)**Description:**

- Gets the TIM Counter Register value on runtime.

Parameters:

- `_HANDLE_`: TIM handle.

Return value:

- None

`_HAL_TIM_SET_AUTORELOAD`

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_GET_AUTORELOAD`

Description:

- Gets the TIM Autoreload Register value on runtime.

Parameters:

- `_HANDLE_`: TIM handle.

Return value:

- None

`_HAL_TIM_SET_CLOCKDIVISION_N`

Description:

- Sets the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- `_HANDLE_`: TIM handle.
- `_CKD_`: specifies the clock division value. This parameter can be one of the following value:
 - `TIM_CLOCKDIVISION_DIV1`
 - `TIM_CLOCKDIVISION_DIV2`
 - `TIM_CLOCKDIVISION_DIV4`

Return value:

- None

`_HAL_TIM_GET_CLOCKDIVISION_N`

Description:

- Gets the TIM Clock Division value on runtime.

Parameters:

- `_HANDLE_`: TIM handle.

Return value:

- None

`_HAL_TIM_SET_ICPRESCALER`

Description:

- Sets the TIM Input Capture prescaler on

runtime without calling another time

Parameters:

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: : TIM Channels to be configured. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: TIM Channel 1 selected
 - `TIM_CHANNEL_2`: TIM Channel 2 selected
 - `TIM_CHANNEL_3`: TIM Channel 3 selected
 - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `__ICPSC__`: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
 - `TIM_ICPSC_DIV1`: no prescaler
 - `TIM_ICPSC_DIV2`: capture is done once every 2 events
 - `TIM_ICPSC_DIV4`: capture is done once every 4 events
 - `TIM_ICPSC_DIV8`: capture is done once every 8 events

Return value:

- None

`__HAL_TIM_GET_ICPRESCALER`

Description:

- Gets the TIM Input Capture prescaler on runtime.

Parameters:

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: get input capture 1 prescaler value
 - `TIM_CHANNEL_2`: get input capture 2 prescaler value
 - `TIM_CHANNEL_3`: get input capture 3 prescaler value
 - `TIM_CHANNEL_4`: get input capture 4 prescaler value

Return value:

- None

`__HAL_TIM_UPS_ENABLE`

Description:

- Set the Update Request Source (URS) bit of the `TIMx_CR1` register.

Parameters:

- `__HANDLE__`: TIM handle.

Return value:

- None

Notes:

- When the USR bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

`__HAL_TIM_URS_DISABLE`

Description:

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- `__HANDLE__`: TIM handle.

Return value:

- None

Notes:

- When the USR bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): (+) Counter overflow/underflow (+) Setting the UG bit (+) Update generation through the slave mode controller

`__HAL_TIM_SET_CAPTUREPOLARITY`

Description:

- Sets the TIM Capture x input polarity on runtime.

Parameters:

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: TIM Channel 1 selected
 - `TIM_CHANNEL_2`: TIM Channel 2 selected
 - `TIM_CHANNEL_3`: TIM Channel 3 selected
 - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `__POLARITY__`: Polarity for TIx source
 - `TIM_INPUTCHANNELPOLARITY_RISING`: Rising Edge
 - `TIM_INPUTCHANNELPOLARITY_FALLING`: Falling Edge
 - `TIM_INPUTCHANNELPOLARITY_BOTH_EDGES`: Rising and Falling Edge

Return value:

- None

Notes:

- The polarity
TIM_INPUTCHANNELPOLARITY_BOTHEDGE
is not authorized for TIM Channel 4.

TIM Flag Definition

TIM_FLAG_UPDATE
 TIM_FLAG_CC1
 TIM_FLAG_CC2
 TIM_FLAG_CC3
 TIM_FLAG_CC4
 TIM_FLAG_COM
 TIM_FLAG_TRIGGER
 TIM_FLAG_BREAK
 TIM_FLAG_CC1OF
 TIM_FLAG_CC2OF
 TIM_FLAG_CC3OF
 TIM_FLAG_CC4OF

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING
 TIM_ICPOLARITY_FALLING
 TIM_ICPOLARITY_BOTHEDGE

TIM Input Capture Prescaler

TIM_ICPSC_DIV1 TIM_ICPSC_DIV2 TIM_ICPSC_DIV4 TIM_ICPSC_DIV8	Capture performed each time an edge is detected on the capture input Capture performed once every 2 events Capture performed once every 4 events Capture performed once every 8 events
--	---

TIM Input Capture Selection

TIM_ICSELECTION_DIRECTTI TIM_ICSELECTION_INDIRECTTI TIM_ICSELECTION_TRC	TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively TIM Input 1, 2, 3 or 4 is selected to be connected to TRC
---	---

TIM Input Channel Polarity

TIM_INPUTCHANNELPOLARITY_RISING	Polarity for TIx source
---------------------------------	-------------------------

TIM_INPUTCHANNELPOLARITY_FALLING	Polarity for TIx source
TIM_INPUTCHANNELPOLARITY_BOTHEDGE	Polarity for TIx source

TIM Interrupt Definition

TIM_IT_UPDATE
TIM_IT_CC1
TIM_IT_CC2
TIM_IT_CC3
TIM_IT_CC4
TIM_IT_COM
TIM_IT_TRIGGER
TIM_IT_BREAK

TIM Lock level

TIM_LOCKLEVEL_OFF
TIM_LOCKLEVEL_1
TIM_LOCKLEVEL_2
TIM_LOCKLEVEL_3

TIM Master Mode Selection

TIM_TRGO_RESET
TIM_TRGO_ENABLE
TIM_TRGO_UPDATE
TIM_TRGO_OC1
TIM_TRGO_OC1REF
TIM_TRGO_OC2REF
TIM_TRGO_OC3REF
TIM_TRGO_OC4REF

TIM Master Slave Mode

TIM_MASTERSLAVEMODE_ENABLE
TIM_MASTERSLAVEMODE_DISABLE

TIM One Pulse Mode

TIM_OPMODE_SINGLE
TIM_OPMODE_REPETITIVE

TIM OSSI Off State Selection for Idle mode state

TIM_OSSI_ENABLE
TIM_OSSI_DISABLE

TIM OSSR Off State Selection for Run mode state

TIM_OSSR_ENABLE

TIM_OSSR_DISABLE

TIM Output Compare and PWM modes

TIM_OCMODE_TIMING

TIM_OCMODE_ACTIVE

TIM_OCMODE_INACTIVE

TIM_OCMODE_TOGGLE

TIM_OCMODE_PWM1

TIM_OCMODE_PWM2

TIM_OCMODE_FORCED_ACTIVE

TIM_OCMODE_FORCED_INACTIVE

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

TIM_OCIDLESTATE_RESET

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

TIM_OCNIDLESTATE_RESET

TIM Complementary Output Compare Polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

TIM Output Fast State

TIM_OCFAST_DISABLE

TIM_OCFAST_ENABLE

TIM Slave Mode

TIM_SLAVEMODE_DISABLE

TIM_SLAVEMODE_RESET

TIM_SLAVEMODE_GATED

TIM_SLAVEMODE_TRIGGER

TIM_SLAVEMODE_EXTERNAL1

TIM TI1 Input Selection

TIM_TI1SELECTION_CH1

TIM_TI1SELECTION_XORCOMBINATION

TIM Trigger Polarity

TIM_TRIGGERPOLARITY_INVERTED Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_NONINVERTED	Polarity for ETRx trigger sources
TIM_TRIGGERPOLARITY_RISING	Polarity for TIxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_FALLING	Polarity for TIxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_BOTHEDGE	Polarity for TIxFPx or TI1_ED trigger sources

TIM Trigger Prescaler

TIM_TRIGGERPRESCALER_DIV1	No prescaler is used
TIM_TRIGGERPRESCALER_DIV2	Prescaler for External ETR Trigger: Capture performed once every 2 events.
TIM_TRIGGERPRESCALER_DIV4	Prescaler for External ETR Trigger: Capture performed once every 4 events.
TIM_TRIGGERPRESCALER_DIV8	Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection

TIM_TS_ITR0
TIM_TS_ITR1
TIM_TS_ITR2
TIM_TS_ITR3
TIM_TS_TI1F_ED
TIM_TS_TI1FP1
TIM_TS_TI2FP2
TIM_TS_ETRF
TIM_TS_NONE

40 HAL TIM Extension Driver

40.1 TIME Firmware driver registers structures

40.1.1 TIM_HallSensor_InitTypeDef

Data Fields

- *uint32_t IC1Polarity*
- *uint32_t IC1Prescaler*
- *uint32_t IC1Filter*
- *uint32_t Commutation_Delay*

Field Documentation

- *uint32_t TIM_HallSensor_InitTypeDef::IC1Polarity*
Specifies the active edge of the input signal. This parameter can be a value of [*TIM_Input_Capture_Polarity*](#)
- *uint32_t TIM_HallSensor_InitTypeDef::IC1Prescaler*
Specifies the Input Capture Prescaler. This parameter can be a value of [*TIM_Input_Capture_Prescaler*](#)
- *uint32_t TIM_HallSensor_InitTypeDef::IC1Filter*
Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- *uint32_t TIM_HallSensor_InitTypeDef::Commutation_Delay*
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

40.1.2 TIM_MasterConfigTypeDef

Data Fields

- *uint32_t MasterOutputTrigger*
- *uint32_t MasterSlaveMode*

Field Documentation

- *uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger*
Trigger output (TRGO) selection This parameter can be a value of [*TIM_Master_Mode_Selection*](#)
- *uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode*
Master/slave mode selection This parameter can be a value of [*TIM_Master_Slave_Mode*](#)

40.1.3 TIM_BreakDeadTimeConfigTypeDef

Data Fields

- *uint32_t OffStateRunMode*
- *uint32_t OffStateIDLEMode*
- *uint32_t LockLevel*
- *uint32_t DeadTime*
- *uint32_t BreakState*
- *uint32_t BreakPolarity*
- *uint32_t AutomaticOutput*

Field Documentation

- *uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode*
TIM off state in run mode This parameter can be a value of
TIM_OSSR_Off_State_Selection_for_Run_mode_state
- *uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode*
TIM off state in IDLE mode This parameter can be a value of
TIM_OSSI_Off_State_Selection_for_Idle_mode_state
- *uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel*
TIM Lock level This parameter can be a value of *TIM_Lock_level*
- *uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime*
TIM dead Time This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- *uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState*
TIM Break State This parameter can be a value of
TIM_Break_Input_enable_disable
- *uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity*
TIM Break input polarity This parameter can be a value of *TIM_Break_Polarity*
- *uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput*
TIM Automatic Output Enable state This parameter can be a value of
TIM_AOE_Bit_Set_Reset

40.2 TIME Firmware driver API description

40.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
 - Output Compare
 - PWM generation (Edge and Center-aligned Mode)
 - One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

40.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :

- Complementary Output Compare : HAL_TIM_OC_MspInit()
 - Complementary PWM generation : HAL_TIM_PWM_MspInit()
 - Complementary One-pulse mode output : HAL_TIM_OnePulse_MspInit()
 - Hall Sensor output : HAL_TIM_HallSensor_MspInit()
2. Initialize the TIM low level resources :
- Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
 - TIM pins configuration
 - Enable the clock for the TIM GPIOs using the following function:
__HAL_RCC_GPIOx_CLK_ENABLE();
 - Configure these TIM pins in Alternate function mode using
HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
- HAL_TIMEx_HallSensor_Init and HAL_TIMEx_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().

40.2.3 Timer Hall Sensor functions

This section provides functions allowing to:

- Initialize and configure TIM HAL Sensor.
- De-initialize TIM HAL Sensor.
- Start the Hall Sensor Interface.
- Stop the Hall Sensor Interface.
- Start the Hall Sensor Interface and enable interrupts.
- Stop the Hall Sensor Interface and disable interrupts.
- Start the Hall Sensor Interface and enable DMA transfers.
- Stop the Hall Sensor Interface and disable DMA transfers.

This section contains the following APIs:

- [***HAL_TIMEx_HallSensor_Init\(\)***](#)
- [***HAL_TIMEx_HallSensor_DeInit\(\)***](#)
- [***HAL_TIMEx_HallSensor_MspInit\(\)***](#)
- [***HAL_TIMEx_HallSensor_MspDeInit\(\)***](#)
- [***HAL_TIMEx_HallSensor_Start\(\)***](#)
- [***HAL_TIMEx_HallSensor_Stop\(\)***](#)
- [***HAL_TIMEx_HallSensor_Start_IT\(\)***](#)
- [***HAL_TIMEx_HallSensor_Stop_IT\(\)***](#)
- [***HAL_TIMEx_HallSensor_Start_DMA\(\)***](#)
- [***HAL_TIMEx_HallSensor_Stop_DMA\(\)***](#)

40.2.4 Timer Complementary Output Compare functions

This section provides functions allowing to:

- Start the Complementary Output Compare/PWM.
- Stop the Complementary Output Compare/PWM.
- Start the Complementary Output Compare/PWM and enable interrupts.
- Stop the Complementary Output Compare/PWM and disable interrupts.
- Start the Complementary Output Compare/PWM and enable DMA transfers.
- Stop the Complementary Output Compare/PWM and disable DMA transfers.

This section contains the following APIs:

- `HAL_TIMEx_OCN_Start()`
- `HAL_TIMEx_OCN_Stop()`
- `HAL_TIMEx_OCN_Start_IT()`
- `HAL_TIMEx_OCN_Stop_IT()`
- `HAL_TIMEx_OCN_Start_DMA()`
- `HAL_TIMEx_OCN_Stop_DMA()`

40.2.5 Timer Complementary PWM functions

This section provides functions allowing to:

- Start the Complementary PWM.
- Stop the Complementary PWM.
- Start the Complementary PWM and enable interrupts.
- Stop the Complementary PWM and disable interrupts.
- Start the Complementary PWM and enable DMA transfers.
- Stop the Complementary PWM and disable DMA transfers.
- Start the Complementary Input Capture measurement.
- Stop the Complementary Input Capture.
- Start the Complementary Input Capture and enable interrupts.
- Stop the Complementary Input Capture and disable interrupts.
- Start the Complementary Input Capture and enable DMA transfers.
- Stop the Complementary Input Capture and disable DMA transfers.
- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- `HAL_TIMEx_PWMN_Start()`
- `HAL_TIMEx_PWMN_Stop()`
- `HAL_TIMEx_PWMN_Start_IT()`
- `HAL_TIMEx_PWMN_Stop_IT()`
- `HAL_TIMEx_PWMN_Start_DMA()`
- `HAL_TIMEx_PWMN_Stop_DMA()`

40.2.6 Timer Complementary One Pulse functions

This section provides functions allowing to:

- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- [*HAL_TIMEx_OnePulseN_Start\(\)*](#)
- [*HAL_TIMEx_OnePulseN_Stop\(\)*](#)
- [*HAL_TIMEx_OnePulseN_Start_IT\(\)*](#)
- [*HAL_TIMEx_OnePulseN_Stop_IT\(\)*](#)

40.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- Configure timer remapping capabilities.

This section contains the following APIs:

- [*HAL_TIMEx_ConfigCommutationEvent\(\)*](#)
- [*HAL_TIMEx_ConfigCommutationEvent_IT\(\)*](#)
- [*HAL_TIMEx_ConfigCommutationEvent_DMA\(\)*](#)
- [*HAL_TIMEx_MasterConfigSynchronization\(\)*](#)
- [*HAL_TIMEx_ConfigBreakDeadTime\(\)*](#)
- [*HAL_TIMEx_RemapConfig\(\)*](#)

40.2.8 Extension Callbacks functions

This section provides Extension TIM callback functions:

- Timer Commutation callback
- Timer Break callback

This section contains the following APIs:

- [*HAL_TIMEx_CommuationCallback\(\)*](#)
- [*HAL_TIMEx_BreakCallback\(\)*](#)
- [*TIMEx_DMACommuationCplt\(\)*](#)

40.2.9 Extension Peripheral State functions

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

This section contains the following APIs:

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

40.2.10 HAL_TIMEx_HallSensor_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (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"> • htim: : TIM Encoder Interface handle • sConfig: : TIM Hall Sensor configuration structure
Return values	• HAL status

40.2.11 HAL_TIMEx_HallSensor_DelInit



Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_DelInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM Hall Sensor interface.
Parameters	<ul style="list-style-type: none">• htim: : TIM Hall Sensor handle
Return values	<ul style="list-style-type: none">• HAL status

40.2.12 HAL_TIMEx_HallSensor_MspInit

Function Name	void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Hall Sensor MSP.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle
Return values	<ul style="list-style-type: none">• None

40.2.13 HAL_TIMEx_HallSensor_MspDelInit

Function Name	void HAL_TIMEx_HallSensor_MspDelInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Hall Sensor MSP.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle
Return values	<ul style="list-style-type: none">• None

40.2.14 HAL_TIMEx_HallSensor_Start

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)
Function Description	Starts the TIM Hall Sensor Interface.
Parameters	<ul style="list-style-type: none">• htim: : TIM Hall Sensor handle
Return values	<ul style="list-style-type: none">• HAL status

40.2.15 HAL_TIMEx_HallSensor_Stop

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)
Function Description	Stops the TIM Hall sensor Interface.
Parameters	<ul style="list-style-type: none">• htim: : TIM Hall Sensor handle
Return values	<ul style="list-style-type: none">• HAL status

40.2.16 HAL_TIMEx_HallSensor_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)
Function Description	Starts the TIM Hall Sensor Interface in interrupt mode.
Parameters	<ul style="list-style-type: none">• htim: : TIM Hall Sensor handle

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

40.2.17 HAL_TIMEx_HallSensor_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)
---------------	--

Function Description	Stops the TIM Hall Sensor Interface in interrupt mode.
----------------------	--

Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
------------	---

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

40.2.18 HAL_TIMEx_HallSensor_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)
---------------	---

Function Description	Starts the TIM Hall Sensor Interface in DMA mode.
----------------------	---

Parameters	<ul style="list-style-type: none"> • htim: : TIM Hall Sensor handle • pData: : The destination Buffer address. • Length: : The length of data to be transferred from TIM peripheral to memory.
------------	--

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

40.2.19 HAL_TIMEx_HallSensor_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)
---------------	---

Function Description	Stops the TIM Hall Sensor Interface in DMA mode.
----------------------	--

Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
------------	---

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

40.2.20 HAL_TIMEx_OCN_Start

Function Name	HAL_StatusTypeDef HAL_TIMEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
---------------	---

Function Description	Starts the TIM Output Compare signal generation on the complementary output.
----------------------	--

Parameters	<ul style="list-style-type: none"> • htim: : TIM Output Compare handle • Channel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
------------	---

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

40.2.21 HAL_TIMEx_OCN_Stop

Function Name	HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
---------------	--

Function Description	Stops the TIM Output Compare signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

40.2.22 HAL_TIMEx_OCN_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> • htim: : TIM OC handle • Channel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

40.2.23 HAL_TIMEx_OCN_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Output Compare handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

40.2.24 HAL_TIMEx_OCN_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Output Compare handle • Channel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2

	selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected <ul style="list-style-type: none"> • pData: : The source Buffer address. • Length: : The length of data to be transferred from memory to TIM peripheral
Return values	<ul style="list-style-type: none"> • HAL status

40.2.25 HAL_TIMEx_OCN_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Output Compare handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

40.2.26 HAL_TIMEx_PWMN_Start

Function Name	HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

40.2.27 HAL_TIMEx_PWMN_Stop

Function Name	HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

40.2.28 HAL_TIMEx_PWMN_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

40.2.29 HAL_TIMEx_PWMN_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

40.2.30 HAL_TIMEx_PWMN_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM PWM signal generation in DMA mode on the complementary output.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected • pData: : The source Buffer address. • Length: : The length of data to be transferred from memory to TIM peripheral
Return values	<ul style="list-style-type: none"> • HAL status

40.2.31 HAL_TIMEx_PWMN_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM PWM signal generation in DMA mode on the

complementary output.

Parameters	<ul style="list-style-type: none"> htim: : TIM handle Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> HAL status

40.2.32 HAL_TIMEx_OnePulseN_Start

Function Name	HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Starts the TIM One Pulse signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> htim: : TIM One Pulse handle OutputChannel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> HAL status

40.2.33 HAL_TIMEx_OnePulseN_Stop

Function Name	HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Stops the TIM One Pulse signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> htim: : TIM One Pulse handle OutputChannel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> HAL status

40.2.34 HAL_TIMEx_OnePulseN_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.
Parameters	<ul style="list-style-type: none"> htim: : TIM One Pulse handle OutputChannel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> HAL status

40.2.35 HAL_TIMEx_OnePulseN_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.
Parameters	<ul style="list-style-type: none"> • htim: : TIM One Pulse handle • OutputChannel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status

40.2.36 HAL_TIMEx_ConfigCommutationEvent

Function Name	HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
Function Description	Configure the TIM commutation event sequence.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • InputTrigger: : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values: TIM_TS_ITR0: Internal trigger 0 selectedTIM_TS_ITR1: Internal trigger 1 selectedTIM_TS_ITR2: Internal trigger 2 selectedTIM_TS_ITR3: Internal trigger 3 selectedTIM_TS_NONE: No trigger is needed • CommutationSource: : the Commutation Event source This parameter can be one of the following values: TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • : 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.

40.2.37 HAL_TIMEx_ConfigCommutationEvent_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
Function Description	Configure the TIM commutation event sequence with interrupt.

Parameters	<ul style="list-style-type: none"> htim: : TIM handle InputTrigger: : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values: TIM_TS_ITR0: Internal trigger 0 selectedTIM_TS_ITR1: Internal trigger 1 selectedTIM_TS_ITR2: Internal trigger 2 selectedTIM_TS_ITR3: Internal trigger 3 selectedTIM_TS_NONE: No trigger is needed CommutationSource: : the Commutation Event source This parameter can be one of the following values: TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface TimerTIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> : 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.

40.2.38 HAL_TIMEx_ConfigCommutationEvent_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
Function Description	Configure the TIM commutation event sequence with DMA.
Parameters	<ul style="list-style-type: none"> htim: : TIM handle InputTrigger: : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values: TIM_TS_ITR0: Internal trigger 0 selectedTIM_TS_ITR1: Internal trigger 1 selectedTIM_TS_ITR2: Internal trigger 2 selectedTIM_TS_ITR3: Internal trigger 3 selectedTIM_TS_NONE: No trigger is needed CommutationSource: : the Commutation Event source This parameter can be one of the following values: TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface TimerTIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to

Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

- : The user should configure the DMA in his own software, in This function only the COMDE bit is set

40.2.39 HAL_TIMEx_MasterConfigSynchronization

Function Name	HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)
Function Description	Configures the TIM in master mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle. • sMasterConfig: : pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.
Return values	<ul style="list-style-type: none"> • HAL status

40.2.40 HAL_TIMEx_ConfigBreakDeadTime

Function Name	HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)
Function Description	Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • sBreakDeadTimeConfig: : pointer to a TIM_BreakDeadTimeConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.
Return values	<ul style="list-style-type: none"> • HAL status

40.2.41 HAL_TIMEx_RemapConfig

Function Name	HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)
Function Description	Configures the TIM14 Remapping input capabilities.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle. • Remap: : specifies the TIM remapping source. This parameter can be one of the following values: TIM_TIM14_GPIO: TIM14 TI1 is connected to GPIO TIM14_RTC: TIM14 TI1 is connected to RTC_clock TIM14_HSE: TIM14 TI1 is connected to HSE/32 TIM14_MCO: TIM14 TI1 is connected to MCO
Return values	<ul style="list-style-type: none"> • HAL status

40.2.42 HAL_TIMEx_CommunicationCallback

Function Name	void HAL_TIMEx_CommunicationCallback (TIM_HandleTypeDef
---------------	--

*** htim)**

Function Description	Hall commutation changed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

40.2.43 HAL_TIMEx_BreakCallback

Function Name	void HAL_TIMEx_BreakCallback (TIM_HandleTypeDef * htim)
Function Description	Hall Break detection callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • None

40.2.44 TIMEx_DMACommutationCplt

Function Name	void TIMEx_DMACommutationCplt (DMA_HandleTypeDef * hdma)
Function Description	TIM DMA Commutation callback.
Parameters	<ul style="list-style-type: none"> • hdma: : pointer to DMA handle.
Return values	<ul style="list-style-type: none"> • None

40.2.45 HAL_TIMEx_HallSensor_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Hall Sensor interface state.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Hall Sensor handle
Return values	<ul style="list-style-type: none"> • HAL state

40.3 TIMEx Firmware driver defines**40.3.1 TIMEx*****TIMEx Clock Filter***

IS_TIM_DEADTIME BreakDead Time

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

41 HAL TSC Generic Driver

41.1 TSC Firmware driver registers structures

41.1.1 TSC_InitTypeDef

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

41.1.2 TSC_IOConfigTypeDef

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

41.1.3 TSC_HandleTypeDef

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

41.2 TSC Firmware driver API description

41.2.1 TSC specific features

- Proven and robust surface charge transfer acquisition principle
- Supports up to 3 capacitive sensing channels per group
- Capacitive sensing channels can be acquired in parallel offering a very good response time
- Spread spectrum feature to improve system robustness in noisy environments
- Full hardware management of the charge transfer acquisition sequence
- Programmable charge transfer frequency
- Programmable sampling capacitor I/O pin
- Programmable channel I/O pin
- Programmable max count value to avoid long acquisition when a channel is faulty
- Dedicated end of acquisition and max count error flags with interrupt capability
- One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
- Compatible with proximity, touchkey, linear and rotary touch sensor implementation

41.2.2 How to use this driver

1. Enable the TSC interface clock using __HAL_RCC_TSC_CLK_ENABLE() macro.
2. GPIO pins configuration
 - Enable the clock for the TSC GPIOs using __HAL_RCC_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.

41.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the TSC.
- De-initialize the TSC.

This section contains the following APIs:

- [*HAL_TSC_Init\(\)*](#)
- [*HAL_TSC_DelInit\(\)*](#)
- [*HAL_TSC_MspInit\(\)*](#)
- [*HAL_TSC_MspDelInit\(\)*](#)

41.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure TSC IOs
- Discharge TSC IOs

This section contains the following APIs:

- [*HAL_TSC_IOConfig\(\)*](#)
- [*HAL_TSC_IODischarge\(\)*](#)

41.2.5 State functions

This subsection provides functions allowing to

- Get TSC state.
- Poll for acquisition completed.
- Handles TSC interrupt request.

This section contains the following APIs:

- [*HAL_TSC_GetState\(\)*](#)
- [*HAL_TSC_PollForAcquisition\(\)*](#)
- [*HAL_TSC_IRQHandler\(\)*](#)

41.2.6 HAL_TSC_Init

Function Name **`HAL_StatusTypeDef HAL_TSC_Init (TSC_HandleTypeDef *htsc)`**

Function Description Initializes the TSC peripheral according to the specified parameters in the `TSC_InitTypeDef` structure.

Parameters • **htsc:** TSC handle

Return values • HAL status

41.2.7 HAL_TSC_DelInit

Function Name **`HAL_StatusTypeDef HAL_TSC_DelInit (TSC_HandleTypeDef *htsc)`**

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

Parameters • **htsc:** TSC handle

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

41.2.8 HAL_TSC_MspInit

Function Name	void HAL_TSC_MspInit (TSC_HandleTypeDef * htsc)
Function Description	Initializes the TSC MSP.
Parameters	<ul style="list-style-type: none"> • htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> • None

41.2.9 HAL_TSC_MspDelInit

Function Name	void HAL_TSC_MspDelInit (TSC_HandleTypeDef * htsc)
Function Description	DeInitializes the TSC MSP.
Parameters	<ul style="list-style-type: none"> • htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> • None

41.2.10 HAL_TSC_Start

Function Name	HAL_StatusTypeDef HAL_TSC_Start (TSC_HandleTypeDef * htsc)
Function Description	Starts the acquisition.
Parameters	<ul style="list-style-type: none"> • htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> • HAL status

41.2.11 HAL_TSC_Start_IT

Function Name	HAL_StatusTypeDef HAL_TSC_Start_IT (TSC_HandleTypeDef * htsc)
Function Description	Enables the interrupt and starts the acquisition.
Parameters	<ul style="list-style-type: none"> • htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> • HAL status.

41.2.12 HAL_TSC_Stop

Function Name	HAL_StatusTypeDef HAL_TSC_Stop (TSC_HandleTypeDef * htsc)
Function Description	Stops the acquisition previously launched in polling mode.
Parameters	<ul style="list-style-type: none"> • htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> • HAL status

41.2.13 HAL_TSC_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TSC_Stop_IT (TSC_HandleTypeDef * htsc)
Function Description	Stops the acquisition previously launched in interrupt mode.
Parameters	<ul style="list-style-type: none"> htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> HAL status

41.2.14 HAL_TSC_GroupGetStatus

Function Name	TSC_GroupStatusTypeDef HAL_TSC_GroupGetStatus (TSC_HandleTypeDef * htsc, uint32_t gx_index)
Function Description	Gets the acquisition status for a group.
Parameters	<ul style="list-style-type: none"> htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. gx_index: Index of the group
Return values	<ul style="list-style-type: none"> Group status

41.2.15 HAL_TSC_GroupGetValue

Function Name	uint32_t HAL_TSC_GroupGetValue (TSC_HandleTypeDef * htsc, uint32_t gx_index)
Function Description	Gets the acquisition measure for a group.
Parameters	<ul style="list-style-type: none"> htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. gx_index: Index of the group
Return values	<ul style="list-style-type: none"> Acquisition measure

41.2.16 HAL_TSC_IOConfig

Function Name	HAL_StatusTypeDef HAL_TSC_IOConfig (TSC_HandleTypeDef * htsc, TSC_IOConfigTypeDef * config)
Function Description	Configures TSC IOs.
Parameters	<ul style="list-style-type: none"> htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. config: pointer to the configuration structure.
Return values	<ul style="list-style-type: none"> HAL status

41.2.17 HAL_TSC_IODischarge

Function Name	HAL_StatusTypeDef HAL_TSC_IODischarge (TSC_HandleTypeDef * htsc, uint32_t choice)
Function Description	Discharge TSC IOs.
Parameters	<ul style="list-style-type: none"> htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. choice: enable or disable
Return values	<ul style="list-style-type: none"> HAL status

41.2.18 HAL_TSC_GetState

Function Name	HAL_TSC_StateTypeDef HAL_TSC_GetState (TSC_HandleTypeDef * htsc)
Function Description	Return the TSC state.
Parameters	<ul style="list-style-type: none"> • htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> • HAL state

41.2.19 HAL_TSC_PollForAcquisition

Function Name	HAL_StatusTypeDef HAL_TSC_PollForAcquisition (TSC_HandleTypeDef * htsc)
Function Description	Start acquisition and wait until completion.
Parameters	<ul style="list-style-type: none"> • htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> • HAL state
Notes	<ul style="list-style-type: none"> • There is no need of a timeout parameter as the max count error is already managed by the TSC peripheral.

41.2.20 HAL_TSC_IRQHandler

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

41.2.21 HAL_TSC_ConvCpltCallback

Function Name	void HAL_TSC_ConvCpltCallback (TSC_HandleTypeDef * htsc)
Function Description	Acquisition completed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> • None

41.2.22 HAL_TSC_ErrorCallback

Function Name	void HAL_TSC_ErrorCallback (TSC_HandleTypeDef * htsc)
Function Description	Error callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	<ul style="list-style-type: none"> • None

41.3 TSC Firmware driver defines

41.3.1 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_OUTPPLOW`

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`

Description:

- Disable TSC interrupt.

Parameters:

- __HANDLE__: TSC handle
- __INTERRUPT__: TSC interrupt

Return value:

- None

__HAL_TSC_GET_IT_SOURCE

Description:

- 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

Description:

- Get the selected TSC's flag status.

Parameters:

- __HANDLE__: TSC handle
- __FLAG__: TSC flag

Return value:

- SET: or RESET

__HAL_TSC_CLEAR_FLAG

Description:

- Clear the TSC's pending flag.

Parameters:

- __HANDLE__: TSC handle
- __FLAG__: TSC flag

Return value:

- None

__HAL_TSC_ENABLE_HYSTERESIS

Description:

- Enable schmitt trigger hysteresis on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_DISABLE_HYSTERESIS

Description:

- Disable schmitt trigger hysteresis on a group of IOs.

Parameters:

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

Return value:

- None

`__HAL_TSC_OPEN_ANALOG_SWITCH`

Description:

- Open analog switch on a group of IOs.

Parameters:

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

Return value:

- None

`__HAL_TSC_CLOSE_ANALOG_SWITCH`

Description:

- Close analog switch on a group of IOs.

Parameters:

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

Return value:

- None

`__HAL_TSC_ENABLE_CHANNEL`

Description:

- Enable a group of IOs in channel mode.

Parameters:

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

Return value:

- None

`__HAL_TSC_DISABLE_CHANNEL`

Description:

- Disable a group of channel IOs.

Parameters:

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

Return value:

- None

`__HAL_TSC_ENABLE_SAMPLING`

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`

- Disable a group of sampling IOs.

Parameters:

- `__HANDLE__`: TSC handle
- `__GX_IOY_MASK__`: IOs mask

Return value:

- None

`__HAL_TSC_ENABLE_GROUP`

- Enable acquisition groups.

Parameters:

- `__HANDLE__`: TSC handle
- `__GX_MASK__`: Groups mask

Return value:

- None

`__HAL_TSC_DISABLE_GROUP`

- Disable acquisition groups.

Parameters:

- `__HANDLE__`: TSC handle
- `__GX_MASK__`: Groups mask

Return value:

- None

`__HAL_TSC_GET_GROUP_STATUS`

- 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_POLARITY_FALLING

TSC_SYNC_POLARITY_RISING

IS_TSC_SYNC_POL

42 HAL UART Generic Driver

42.1 UART Firmware driver registers structures

42.1.1 **UART_InitTypeDef**

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 f_PCLK/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***.

42.1.2 **UART_AdvFeatureInitTypeDef**

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**](#)

42.1.3 **UART_HandleTypeDef**

Data Fields

- **USART_TypeDef * Instance**
- **UART_InitTypeDef Init**
- **UART_AdvFeatureInitTypeDef AdvancedInit**
- **uint8_t * pTxBuffPtr**
- **uint16_t TxXferSize**
- **uint16_t TxXferCount**
- **uint8_t * pRxBuffPtr**
- **uint16_t RxXferSize**
- **uint16_t RxXferCount**
- **uint16_t Mask**
- **DMA_HandleTypeDef * hdmatx**
- **DMA_HandleTypeDef * hdmarx**
- **HAL_LockTypeDef Lock**
- **__IO HAL_UART_StateTypeDef State**
- **__IO uint32_t ErrorCode**

Field Documentation

- **USART_TypeDef* UART_HandleTypeDef::Instance**
UART registers base address
- **UART_InitTypeDef UART_HandleTypeDef::Init**
UART communication parameters
- **UART_AdvFeatureInitTypeDef UART_HandleTypeDef::AdvancedInit**
UART Advanced Features initialization parameters
- **uint8_t* UART_HandleTypeDef::pTxBuffPtr**
Pointer to UART Tx transfer Buffer
- **uint16_t UART_HandleTypeDef::TxXferSize**
UART Tx Transfer size
- **uint16_t UART_HandleTypeDef::TxXferCount**
UART Tx Transfer Counter
- **uint8_t* UART_HandleTypeDef::pRxBuffPtr**
Pointer to UART Rx transfer Buffer
- **uint16_t UART_HandleTypeDef::RxXferSize**
UART Rx Transfer size
- **uint16_t UART_HandleTypeDef::RxXferCount**
UART Rx Transfer Counter
- **uint16_t UART_HandleTypeDef::Mask**
UART Rx RDR register mask
- **DMA_HandleTypeDef* UART_HandleTypeDef::hdmatx**
UART Tx DMA Handle parameters
- **DMA_HandleTypeDef* UART_HandleTypeDef::hdmarx**
UART Rx DMA Handle parameters
- **HAL_LockTypeDef UART_HandleTypeDef::Lock**
Locking object
- **__IO HAL_UART_StateTypeDef UART_HandleTypeDef::State**
UART communication state
- **__IO uint32_t UART_HandleTypeDef::ErrorCode**
UART Error code

42.2 UART Firmware driver API description

42.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a UART_HandleTypeDef handle structure (eg. UART_HandleTypeDef huart).
2. Initialize the UART low level resources by implementing the HAL_UART_MspInit() API:
 - Enable the USARTx interface clock.
 - UART pins configuration:
 - Enable the clock for the UART GPIOs.
 - Configure these UART pins as alternate function pull-up.
 - 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.
 - UART interrupts handling: The specific UART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) are managed using the macros __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive processes.
 - 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 handle Init structure.
4. If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...) in the huart handle 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.



These APIs(HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_MultiProcessor_Init(), also configure 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_DMAMstop()

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

42.2.2 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 the parameters below 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. According to device capability (support or not of 7-bit word length), frame length is either defined by the M bit (8-bits or 9-bits) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit). Possible USART frame formats are as listed in the table below.
 - 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: 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
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 USART asynchronous, USART Half duplex and multiprocessor mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [*HAL_UART_Init\(\)*](#)
- [*HAL_HalfDuplex_Init\(\)*](#)
- [*HAL_MultiProcessor_Init\(\)*](#)
- [*HAL_UART_DeInit\(\)*](#)
- [*HAL_UART_MspInit\(\)*](#)
- [*HAL_UART_MspDeInit\(\)*](#)

42.2.3 IO operation functions

This section contains the following APIs:

- [*HAL_UART_Transmit\(\)*](#)
- [*HAL_UART_Receive\(\)*](#)
- [*HAL_UART_Transmit_IT\(\)*](#)
- [*HAL_UART_Receive_IT\(\)*](#)
- [*HAL_UART_Transmit_DMA\(\)*](#)
- [*HAL_UART_Receive_DMA\(\)*](#)
- [*HAL_UART_DMAPause\(\)*](#)
- [*HAL_UART_DMAResume\(\)*](#)
- [*HAL_UART_DMAStop\(\)*](#)
- [*HAL_UART_TxCpltCallback\(\)*](#)
- [*HAL_UART_TxHalfCpltCallback\(\)*](#)
- [*HAL_UART_RxCpltCallback\(\)*](#)
- [*HAL_UART_RxHalfCpltCallback\(\)*](#)
- [*HAL_UART_ErrorCallback\(\)*](#)
- [*HAL_UART_IRQHandler\(\)*](#)

42.2.4 Peripheral Control functions

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

- [*HAL_MultiProcessor_EnableMuteMode\(\)*](#) API enables mute mode
- [*HAL_MultiProcessor_DisableMuteMode\(\)*](#) API disables mute mode
- [*HAL_MultiProcessor_EnterMuteMode\(\)*](#) API enters mute mode
- [*HAL_HalfDuplex_EnableTransmitter\(\)*](#) API disables receiver and enables transmitter
- [*HAL_HalfDuplex_EnableReceiver\(\)*](#) API disables transmitter and enables receiver

This section contains the following APIs:

- [*HAL_MultiProcessor_EnableMuteMode\(\)*](#)
- [*HAL_MultiProcessor_DisableMuteMode\(\)*](#)
- [*HAL_MultiProcessor_EnterMuteMode\(\)*](#)
- [*HAL_HalfDuplex_EnableTransmitter\(\)*](#)
- [*HAL_HalfDuplex_EnableReceiver\(\)*](#)

42.2.5 Peripheral State and Error functions

This subsection provides functions allowing to :

- Return the UART handle state.
- Return the UART handle error code

This section contains the following APIs:

- [*HAL_UART_GetState\(\)*](#)
- [*HAL_UART_GetError\(\)*](#)

42.2.6 HAL_UART_Init

Function Name	HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)
Function Description	Initialize the UART mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.
Parameters	<ul style="list-style-type: none"> • huart: UART handle.
Return values	<ul style="list-style-type: none"> • HAL status

42.2.7 HAL_HalfDuplex_Init

Function Name	HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)
Function Description	Initialize the half-duplex mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • huart: UART handle.
Return values	<ul style="list-style-type: none"> • HAL status

42.2.8 HAL_MultiProcessor_Init

Function Name	HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)
Function Description	Initialize the multiprocessor mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.
Parameters	<ul style="list-style-type: none"> • huart: UART handle. • Address: UART node address (4-, 6-, 7- or 8-bit long). • WakeUpMethod: specifies the UART wakeup method. This parameter can be one of the following values: UART_WAKEUPMETHOD_IDLELINE: WakeUp by an idle line detection UART_WAKEUPMETHOD_ADDRESSMARK: WakeUp by an address mark
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function. • If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API HAL_MultiProcessorEx_AddressLength_Set() must be called after HAL_MultiProcessor_Init().

42.2.9 HAL_UART_DeInit

Function Name	HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)
Function Description	Deinitialize the UART peripheral.

Parameters	<ul style="list-style-type: none"> huart: UART handle.
Return values	<ul style="list-style-type: none"> HAL status

42.2.10 HAL_UART_MspInit

Function Name	void HAL_UART_MspInit (UART_HandleTypeDef * huart)
Function Description	Initialize the UART MSP.
Parameters	<ul style="list-style-type: none"> huart: UART handle.
Return values	<ul style="list-style-type: none"> None

42.2.11 HAL_UART_MspDeInit

Function Name	void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)
Function Description	Deinitialize the UART MSP.
Parameters	<ul style="list-style-type: none"> huart: UART handle.
Return values	<ul style="list-style-type: none"> None

42.2.12 HAL_UART_Transmit

Function Name	HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> huart: UART handle. pData: Pointer to data buffer. Size: Amount of data to be sent. Timeout: Timeout duration.
Return values	<ul style="list-style-type: none"> HAL status

42.2.13 HAL_UART_Receive

Function Name	HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> huart: UART handle. pData: pointer to data buffer. Size: amount of data to be received. Timeout: Timeout duration.
Return values	<ul style="list-style-type: none"> HAL status

42.2.14 HAL_UART_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in interrupt mode.

Parameters	<ul style="list-style-type: none"> huart: UART handle. pData: pointer to data buffer. Size: amount of data to be sent.
Return values	<ul style="list-style-type: none"> HAL status

42.2.15 HAL_UART_Receive_IT

Function Name	HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> huart: UART handle. pData: pointer to data buffer. Size: amount of data to be received.
Return values	<ul style="list-style-type: none"> HAL status

42.2.16 HAL_UART_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> huart: UART handle. pData: pointer to data buffer. Size: amount of data to be sent.
Return values	<ul style="list-style-type: none"> HAL status

42.2.17 HAL_UART_Receive_DMA

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

42.2.18 HAL_UART_DMAPause

Function Name	HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)
Function Description	Pause the DMA Transfer.
Parameters	<ul style="list-style-type: none"> huart: UART handle.
Return values	<ul style="list-style-type: none"> HAL status

42.2.19 HAL_UART_DMAResume

Function Name	HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)
Function Description	Resume the DMA Transfer.
Parameters	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• HAL status

42.2.20 HAL_UART_DMAStop

Function Name	HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)
Function Description	Stop the DMA Transfer.
Parameters	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• HAL status

42.2.21 HAL_UART_TxCpltCallback

Function Name	void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)
Function Description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• None

42.2.22 HAL_UART_TxHalfCpltCallback

Function Name	void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)
Function Description	Tx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• None

42.2.23 HAL_UART_RxCpltCallback

Function Name	void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)
Function Description	Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none">• huart: UART handle.
Return values	<ul style="list-style-type: none">• None

42.2.24 HAL_UART_RxHalfCpltCallback

Function Name	void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)
Function Description	Rx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none">• huart: UART handle.

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

42.2.25 HAL_UART_ErrorCallback

Function Name	<code>void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)</code>
Function Description	UART error callback.
Parameters	<ul style="list-style-type: none"> huart: UART handle.
Return values	<ul style="list-style-type: none"> None

42.2.26 HAL_UART_IRQHandler

Function Name	<code>void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)</code>
Function Description	Handle UART interrupt request.
Parameters	<ul style="list-style-type: none"> huart: UART handle.
Return values	<ul style="list-style-type: none"> None

42.2.27 HAL_MultiProcessor_EnableMuteMode

Function Name	<code>HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)</code>
Function Description	Enable UART in mute mode (does not 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"> huart: UART handle.
Return values	<ul style="list-style-type: none"> HAL status

42.2.28 HAL_MultiProcessor_DisableMuteMode

Function Name	<code>HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)</code>
Function Description	Disable UART mute mode (does not mean the UART actually exits mute mode as it may not have been in mute mode at this very moment).
Parameters	<ul style="list-style-type: none"> huart: UART handle.
Return values	<ul style="list-style-type: none"> HAL status

42.2.29 HAL_MultiProcessor_EnterMuteMode

Function Name	<code>void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)</code>
Function Description	Enter UART mute mode (means UART actually enters mute mode).
Parameters	<ul style="list-style-type: none"> huart: UART handle.
Return values	<ul style="list-style-type: none"> None

Notes

- To exit from mute mode,
HAL_MultiProcessor_DisableMuteMode() API must be called.

42.2.30 HAL_HalfDuplex_EnableTransmitter

Function Name	HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)
Function Description	Enable the UART transmitter and disable the UART receiver.
Parameters	<ul style="list-style-type: none"> • huart: UART handle.
Return values	<ul style="list-style-type: none"> • HAL status

42.2.31 HAL_HalfDuplex_EnableReceiver

Function Name	HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)
Function Description	Enable the UART receiver and disable the UART transmitter.
Parameters	<ul style="list-style-type: none"> • huart: UART handle.
Return values	<ul style="list-style-type: none"> • HAL status.

42.2.32 HAL_UART_GetState

Function Name	HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)
Function Description	Return the UART handle state.
Parameters	<ul style="list-style-type: none"> • huart: : pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	<ul style="list-style-type: none"> • HAL state

42.2.33 HAL_UART_GetError

Function Name	uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)
Function Description	Return the UART handle error code.
Parameters	<ul style="list-style-type: none"> • huart: : pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	<ul style="list-style-type: none"> • UART Error Code

42.3 UART Firmware driver defines**42.3.1 UART*****UART Advanced Feature Initialization Type***

UART_ADVFEATURE_NO_INIT	No advanced feature initialization
UART_ADVFEATURE_TXINVERT_INIT	TX pin active level inversion
UART_ADVFEATURE_RXINVERT_INIT	RX pin active level inversion
UART_ADVFEATURE_DATAINVERT_INIT	Binary data inversion

UART_ADVFEATURE_SWAP_INIT	TX/RX pins swap
UART_ADVFEATURE_RXOVERRUNDISABLE_INIT	RX overrun disable
UART_ADVFEATURE_DMADISABLEONERROR_INIT	DMA disable on Reception Error
UART_ADVFEATURE_AUTOBAUDRATE_INIT	Auto Baud rate detection initialization
UART_ADVFEATURE_MSBFIRST_INIT	Most significant bit sent/received first
UART Advanced Feature Auto BaudRate Enable	
UART_ADVFEATURE_AUTOBAUDRATE_DISABLE	RX Auto Baud rate detection enable
UART_ADVFEATURE_AUTOBAUDRATE_ENABLE	RX Auto Baud rate detection disable
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	Binary data inversion disable
UART_ADVFEATURE_DATAINV_ENABLE	Binary data inversion enable
UART Advanced Feature DMA Disable On Rx Error	
UART_ADVFEATURE_DMA_ENABLEONRXERROR	DMA enable on Reception Error
UART_ADVFEATURE_DMA_DISABLEONRXERROR	DMA disable on Reception Error
UART DMA Rx	
UART_DMA_RX_DISABLE	UART DMA RX disabled
UART_DMA_RX_ENABLE	UART DMA RX enabled
UART DMA Tx	
UART_DMA_TX_DISABLE	UART DMA TX disabled
UART_DMA_TX_ENABLE	UART DMA TX enabled
UART DriverEnable Polarity	
UART_DE_POLARITY_HIGH	Driver enable signal is active high
UART_DE_POLARITY_LOW	Driver enable signal is active low
UART Error	
HAL_UART_ERROR_NONE	No error
HAL_UART_ERROR_PE	Parity error
HAL_UART_ERROR_NE	Noise error
HAL_UART_ERROR_FE	frame error

HAL_UART_ERROR_ORE	Overrun error
HAL_UART_ERROR_DMA	DMA transfer error

UART Exported Macros

__HAL_UART_RESET_HANDLE_STAT	Description: <ul style="list-style-type: none">Reset UART handle state. Parameters: <ul style="list-style-type: none"><code>__HANDLE__</code>: UART handle. Return value: <ul style="list-style-type: none">None
__HAL_UART_CLEAR_FLAG	Description: <ul style="list-style-type: none">Clear the specified UART pending flag. Parameters: <ul style="list-style-type: none"><code>__HANDLE__</code>: specifies the UART Handle.<code>__FLAG__</code>: specifies the flag to check. This parameter can be any combination of the following values:<ul style="list-style-type: none"><code>UART_CLEAR_PEF</code>, Parity Error Clear Flag<code>UART_CLEAR_FEF</code>, Framing Error Clear Flag<code>UART_CLEAR_NEF</code>, Noise detected Clear Flag<code>UART_CLEAR_OREF</code>, OverRun Error Clear Flag<code>UART_CLEAR_IDLEF</code>, IDLE line detected Clear Flag<code>UART_CLEAR_TCF</code>, Transmission Complete Clear Flag<code>UART_CLEAR_LBDF</code>, LIN Break Detection Clear Flag (not available on all devices)<code>UART_CLEAR_CTSF</code>, CTS Interrupt Clear Flag<code>UART_CLEAR_RTOF</code>, Receiver Time Out Clear Flag<code>UART_CLEAR_EOBF</code>, End Of Block Clear Flag (not available on all devices)<code>UART_CLEAR_CMF</code>, Character Match Clear Flag<code>UART_CLEAR_WUF</code>, Wake Up from stop mode Clear Flag (not available on all devices) Return value: <ul style="list-style-type: none">None

`__HAL_UART_CLEAR_PEFLAG`

Description:

- Clear the UART PE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

`__HAL_UART_CLEAR_FEFLAG`

Description:

- Clear the UART FE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

`__HAL_UART_CLEAR_NEFLAG`

Description:

- Clear the UART NE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

`__HAL_UART_CLEAR_OREFLAG`

Description:

- Clear the UART ORE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

`__HAL_UART_CLEAR_IDLEFLAG`

Description:

- Clear the UART IDLE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

`__HAL_UART_GET_FLAG`

Description:

- Check whether the specified UART flag is 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:

- Enable the specified UART interrupt.

Parameters:

- _HANDLE_: specifies the UART Handle. This parameter can be USART_x 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:

- Disable the specified UART interrupt.

Parameters:

- _HANDLE_: specifies the UART Handle. This parameter can be USART_x 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

- Check whether the specified UART interrupt has occurred or not.

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)
- _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

Return value:

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

`__HAL_UART_GET_IT_SOURCE`**Description:**

- Check whether the specified UART interrupt source is enabled 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 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:**

- Clear the specified UART ISR flag, in setting the proper ICR register 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)
- [__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](#)**Description:**

- 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_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the UART one bit sample method.

Parameters:

- HANDLE: specifies the UART Handle.

Return value:

- None

_HAL_UART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the UART one bit sample method.

Parameters:

- HANDLE: specifies the UART Handle.

Return value:

- None

_HAL_UART_ENABLE

Description:

- Enable UART.

Parameters:

- HANDLE: specifies the UART Handle. The Handle Instance can be USARTx where x: 1, 2, 3, 4 or 5 to select the USART 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

`__HAL_UART_HWCONTROL_CTS_EN
ABLE`

Description:

- Enable CTS flow control.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable CTS hardware flow control for a given UART instance, without need to call `HAL_UART_Init()` function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of `HAL_UART_Init()`) macro could only be called when corresponding UART instance is disabled (i.e. `__HAL_UART_DISABLE(__HANDLE__)`) and should be followed by an Enable macro (i.e. `__HAL_UART_ENABLE(__HANDLE__)`).

`__HAL_UART_HWCONTROL_CTS_DIS
ABLE`

Description:

- Disable CTS flow control.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable CTS hardware flow control for a given UART

instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : USART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding USART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_ENABLE

Description:

- Enable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

• This macro allows to enable RTS hardware flow control for a given USART instance, without need to call HAL_UART_Init() function. As involving direct access to USART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : USART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding USART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_DISABLE

Description:

- Disable RTS flow control.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable RTS hardware flow control for a given UART instance, without need to call `HAL_UART_Init()` function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of `HAL_UART_Init()`) macro could only be called when corresponding UART instance is disabled (i.e. `__HAL_UART_DISABLE(__HANDLE__)`) and should be followed by an Enable macro (i.e. `__HAL_UART_ENABLE(__HANDLE__)`).

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 disabled
UART_HALF_DUPLEX_ENABLE UART half-duplex enabled

UART Hardware Flow Control

UART_HWCONTROL_NONE No hardware control
UART_HWCONTROL_RTS Request To Send
UART_HWCONTROL_CTS Clear To Send
UART_HWCONTROL_RTS_CTS Request and Clear To Send

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	RX mode
UART_MODE_TX	TX mode
UART_MODE_TX_RX	RX and TX mode
UART Advanced Feature MSB First	
UART_ADVFEATURE_MSBFIRST_DISABLE	Most significant bit sent/received first disable
UART_ADVFEATURE_MSBFIRST_ENABLE	Most significant bit sent/received first enable
UART Advanced Feature Mute Mode Enable	
UART_ADVFEATURE_MUTEMODE_DISABLE	UART mute mode disable
UART_ADVFEATURE_MUTEMODE_ENABLE	UART mute mode enable
UART One Bit Sampling Method	
UART_ONE_BIT_SAMPLE_DISABLE	One-bit sampling disable
UART_ONE_BIT_SAMPLE_ENABLE	One-bit sampling enable
UART Advanced Feature Overrun Disable	
UART_ADVFEATURE_OVERRUN_ENABLE	RX overrun enable
UART_ADVFEATURE_OVERRUN_DISABLE	RX overrun disable
UART Over Sampling	
UART_OVERSAMPLING_16	Oversampling by 16
UART_OVERSAMPLING_8	Oversampling by 8
UART Parity	
UART_PARITY_NONE	No parity
UART_PARITY EVEN	Even parity
UART_PARITY_ODD	Odd parity
UART Receiver TimeOut	
UART_RECEIVER_TIMEOUT_DISABLE	UART receiver timeout disable
UART_RECEIVER_TIMEOUT_ENABLE	UART receiver timeout enable
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
UART Advanced Feature RX Pin Active Level Inversion	
UART_ADVFEATURE_RXINV_DISABLE	RX pin active level inversion disable
UART_ADVFEATURE_RXINV_ENABLE	RX pin active level inversion enable
UART Advanced Feature RX TX Pins Swap	
UART_ADVFEATURE_SWAP_DISABLE	TX/RX pins swap disable
UART_ADVFEATURE_SWAP_ENABLE	TX/RX pins swap enable
UART State	
UART_STATE_DISABLE	UART disabled
UART_STATE_ENABLE	UART enabled
UART Number of Stop Bits	
UART_STOPBITS_1	
UART_STOPBITS_2	
UART_STOPBITS_1_5	
UARTEx Advanced Feature Stop Mode Enable	
UART_ADVFEATURE_STOPMODE_DISABLE	UART stop mode disable
UART_ADVFEATURE_STOPMODE_ENABLE	UART stop mode enable
UART polling-based communications time-out value	
HAL_UART_TIMEOUT_VALUE	
UART Advanced Feature TX Pin Active Level Inversion	
UART_ADVFEATURE_TXINV_DISABLE	TX pin active level inversion disable
UART_ADVFEATURE_TXINV_ENABLE	TX pin active level inversion enable
UART WakeUp Address Length	
UART_ADDRESS_DETECT_4B	4-bit long wake-up address
UART_ADDRESS_DETECT_7B	7-bit long wake-up address
UART WakeUp From Stop Selection	
UART_WAKEUP_ON_ADDRESS	UART wake-up on address
UART_WAKEUP_ON_STARTBIT	UART wake-up on start bit
UART_WAKEUP_ON_READDATA_NONEMPTY	UART wake-up on receive data register not empty
UART WakeUp Methods	
UART_WAKEUPMETHOD_IDLELINE	UART wake-up on idle line
UART_WAKEUPMETHOD_ADDRESSMARK	UART wake-up on address mark

43 HAL UART Extension Driver

43.1 UARTEEx Firmware driver registers structures

43.1.1 UART_WakeUpTypeDef

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)

43.2 UARTEEx Firmware driver API description

43.2.1 UART peripheral extended features

43.2.2 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 the parameters below 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. According to device capability (support or not of 7-bit word length), frame length is either defined by the M bit (8-bits or 9-bits) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit). Possible UART frame formats are as listed in the table below:
 - 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 24: UART 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
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_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).

This section contains the following APIs:

- [**HAL_RS485Ex_Init\(\)**](#)
- [**HAL_LIN_Init\(\)**](#)

43.2.3 IO operation function

This subsection provides functions allowing to manage the USART 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_UARTEx_WakeupCallback\(\)**](#)

This section contains the following APIs:

- [**HAL_UART_IRQHandler\(\)**](#)
- [**HAL_UARTEx_WakeupCallback\(\)**](#)

43.2.4 Peripheral Control function

This subsection provides extended functions allowing to control the USART.

- [**HAL_UARTEx_StopModeWakeUpSourceConfig\(\)**](#) API sets Wakeup from Stop mode interrupt flag selection

- HAL_UARTEEx_EnableStopMode() API allows the UART to wake up the MCU from Stop mode as long as UART clock is HSI or LSE
- HAL_UARTEEx_DisableStopMode() API disables the above feature
- 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_LIN_SendBreak() API transmits the break characters

This section contains the following APIs:

- [**HAL_UARTEEx_StopModeWakeUpSourceConfig\(\)**](#)
- [**HAL_UARTEEx_EnableStopMode\(\)**](#)
- [**HAL_UARTEEx_DisableStopMode\(\)**](#)
- [**HAL_MultiProcessorEx_AddressLength_Set\(\)**](#)
- [**HAL_LIN_SendBreak\(\)**](#)

43.2.5 HAL_RS485Ex_Init

Function Name	HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)
Function Description	Initialize 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"> • huart: UART handle. • Polarity: select the driver enable polarity. This parameter can be one of the following values: UART_DE_POLARITY_HIGH: DE signal is active highUART_DE_POLARITY_LOW: DE signal is active low • AssertionTime: Driver Enable assertion time: 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate) • DeassertionTime: Driver Enable deassertion time: 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).
Return values	<ul style="list-style-type: none"> • HAL status

43.2.6 HAL_LIN_Init

Function Name	HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)
Function Description	Initialize the LIN mode according to the specified parameters in the <code>UART_InitTypeDef</code> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> • huart: UART handle. • BreakDetectLength: specifies the LIN break detection length. This parameter can be one of the following values: UART_LINBREAKDETECTLENGTH_10B: 10-bit break detection UART_LINBREAKDETECTLENGTH_11B: 11-bit break detection

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

43.2.7 HAL_UART_IRQHandler

Function Name	void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)
Function Description	Handle UART interrupt request.
Parameters	<ul style="list-style-type: none"> • huart: UART handle.
Return values	<ul style="list-style-type: none"> • None

43.2.8 HAL_UARTEx_WakeupCallback

Function Name	void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)
Function Description	UART wakeup from Stop mode callback.
Parameters	<ul style="list-style-type: none"> • huart: UART handle
Return values	<ul style="list-style-type: none"> • None

43.2.9 HAL_UARTEx_StopModeWakeUpSourceConfig

Function Name	HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)
Function Description	Set Wakeup from Stop mode interrupt flag selection.
Parameters	<ul style="list-style-type: none"> • huart: UART handle. • WakeUpSelection: address match, Start Bit detection or RXNE bit status. This parameter can be one of the following values: UART_WAKEUP_ON_ADDRESSUART_WAKEUP_ON_STARTBITUART_WAKEUP_ON_RXADDATA_NONEMPTY
Return values	<ul style="list-style-type: none"> • HAL status

43.2.10 HAL_UARTEx_EnableStopMode

Function Name	HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (UART_HandleTypeDef * huart)
Function Description	Enable UART Stop Mode.
Parameters	<ul style="list-style-type: none"> • huart: UART handle.
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- The UART is able to wake up the MCU from Stop 1 mode as long as UART clock is HSI or LSE.

43.2.11 HAL_UARTEx_DisableStopMode

Function Name	HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (UART_HandleTypeDef * huart)
---------------	--

Function Description	Disable UART Stop Mode.
Parameters	<ul style="list-style-type: none"> • huart: UART handle.
Return values	<ul style="list-style-type: none"> • HAL status

43.2.12 HAL_MultiProcessorEx_AddressLength_Set

Function Name	HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)
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; this API allows to enable longer addresses detection (6-, 7- or 8-bit long).
Parameters	<ul style="list-style-type: none"> • huart: UART handle. • AddressLength: this parameter can be one of the following values: UART_ADDRESS_DETECT_4B: 4-bit long addressUART_ADDRESS_DETECT_7B: 6-, 7- or 8-bit long address
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • Addresses detection lengths are: 6-bit address detection in 7-bit data mode, 7-bit address detection in 8-bit data mode, 8-bit address detection in 9-bit data mode.

43.2.13 HAL_LIN_SendBreak

Function Name	HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)
Function Description	Transmit break characters.
Parameters	<ul style="list-style-type: none"> • huart: UART handle.
Return values	<ul style="list-style-type: none"> • HAL status

43.3 UARTE Firmware driver defines

43.3.1 UARTE

UARTE Advanced Feature AutoBaud Rate Mode

UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT	Auto Baud rate detection on start bit
UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE	Auto Baud rate detection on falling edge
UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFFRAME	Auto Baud rate detection on 0x7F frame detection
UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME	Auto Baud rate detection on 0x55 frame detection

UARTE Exported Macros

_HAL_UART_FLUSH_DRREGISTER Description:

- Flush the UART Data registers.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

UARTEx Local Interconnection Network mode

`UART_LIN_DISABLE` Local Interconnect Network disable

`UART_LIN_ENABLE` Local Interconnect Network enable

UARTEx LIN Break Detection

`UART_LINBREAKDETECTLENGTH_10B` LIN 10-bit break detection length

`UART_LINBREAKDETECTLENGTH_11B` LIN 11-bit break detection length

UARTEx Word Length

`UART_WORDLENGTH_7B`

`UART_WORDLENGTH_8B`

`UART_WORDLENGTH_9B`

44 HAL USART Generic Driver

44.1 USART Firmware driver registers structures

44.1.1 USART_InitTypeDef

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**](#).

44.1.2 USART_HandleTypeDef

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**
- **__IO HAL_USART_StateTypeDef State**
- **__IO uint32_t ErrorCode**

Field Documentation

- **USART_TypeDef* USART_HandleTypeDef::Instance**
USART registers base address
- **USART_InitTypeDef USART_HandleTypeDef::Init**
USART communication parameters
- **uint8_t* USART_HandleTypeDef::pTxBuffPtr**
Pointer to USART Tx transfer Buffer
- **uint16_t USART_HandleTypeDef::TxXferSize**
USART Tx Transfer size
- **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
- **__IO HAL_USART_StateTypeDef USART_HandleTypeDef::State**
USART communication state
- **__IO uint32_t USART_HandleTypeDef::ErrorCode**
USART Error code

44.2 USART Firmware driver API description

44.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure (eg. USART_HandleTypeDef husart).
 2. Initialize the USART low level resources by implementing the HAL_USART_MspInit() API:
 - Enable the USARTTx interface clock.
 - USART pins configuration:
 - Enable the clock for the USART GPIOs.
 - Configure these USART pins as alternate function pull-up.
 - 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 USARTTx interrupt priority.
 - Enable the NVIC USART IRQ handle.
 - USART interrupts handling: 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.
 - 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 handle Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
 - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc by calling the customized HAL_USART_MspInit(&husart) API.
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_DMAMstop()

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

44.2.2 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. According to device capability (support or not of 7-bit word length), frame length is either defined by the M bit (8-bits or 9-bits) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit). Possible USART frame formats are as listed in the table below:
 - USART polarity
 - USART phase
 - USART LastBit
 - Receiver/transmitter modes

Table 25: 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
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_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

This section contains the following APIs:

- [**HAL_USART_Init\(\)**](#)
- [**HAL_USART_DeInit\(\)**](#)
- [**HAL_USART_MspInit\(\)**](#)
- [**HAL_USART_MspDeInit\(\)**](#)

44.2.3 IO operation functions

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

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:

- Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA, These 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. No-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. No-Blocking mode APIs with DMA are :
 - HAL_USART_Transmit_DMA() in simplex mode
 - HAL_USART_Receive_DMA() in full duplex receive only
 - HAL_USART_TransmitReceive_DMA() in full duplex mode
 - HAL_USART_DMAPause()
 - HAL_USART_DMAResume()
 - HAL_USART_DMAStop()
 5. A set of Transfer Complete Callbacks are provided in No-Blocking mode:
 - HAL_USART_TxCpltCallback()
 - HAL_USART_RxCpltCallback()
 - HAL_USART_TxHalfCpltCallback()
 - HAL_USART_RxHalfCpltCallback()
 - HAL_USART_ErrorCallback()
 - HAL_USART_TxRxCpltCallback()

This section contains the following APIs:

- [**HAL_USART_Transmit\(\)**](#)
- [**HAL_USART_Receive\(\)**](#)
- [**HAL_USART_TransmitReceive\(\)**](#)
- [**HAL_USART_Transmit_IT\(\)**](#)
- [**HAL_USART_Receive_IT\(\)**](#)
- [**HAL_USART_TransmitReceive_IT\(\)**](#)
- [**HAL_USART_Transmit_DMA\(\)**](#)
- [**HAL_USART_Receive_DMA\(\)**](#)
- [**HAL_USART_TransmitReceive_DMA\(\)**](#)
- [**HAL_USART_DMAPause\(\)**](#)
- [**HAL_USART_DMAResume\(\)**](#)
- [**HAL_USART_DMAStop\(\)**](#)
- [**HAL_USART_IRQHandler\(\)**](#)
- [**HAL_USART_TxCpltCallback\(\)**](#)
- [**HAL_USART_TxHalfCpltCallback\(\)**](#)
- [**HAL_USART_RxCpltCallback\(\)**](#)
- [**HAL_USART_RxHalfCpltCallback\(\)**](#)

- `HAL_USART_TxRxCpltCallback()`
- `HAL_USART_ErrorCallback()`

44.2.4 Peripheral State and Error functions

This subsection provides functions allowing to :

- Return the USART handle state
- Return the USART handle error code

This section contains the following APIs:

- `HAL_USART_GetState()`
- `HAL_USART_GetError()`

44.2.5 HAL_USART_Init

Function Name	<code>HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * huart)</code>
Function Description	Initializes the USART mode according to the specified parameters in the <code>USART_InitTypeDef</code> and initialize the associated handle.
Parameters	<ul style="list-style-type: none"> • <code>huart</code>: USART handle.
Return values	<ul style="list-style-type: none"> • HAL status

44.2.6 HAL_USART_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * huart)</code>
Function Description	DeInitialize the USART peripheral.
Parameters	<ul style="list-style-type: none"> • <code>huart</code>: USART handle.
Return values	<ul style="list-style-type: none"> • HAL status

44.2.7 HAL_USART_MspInit

Function Name	<code>void HAL_USART_MspInit (USART_HandleTypeDef * huart)</code>
Function Description	Initialize the USART MSP.
Parameters	<ul style="list-style-type: none"> • <code>huart</code>: USART handle.
Return values	<ul style="list-style-type: none"> • None

44.2.8 HAL_USART_MspDeInit

Function Name	<code>void HAL_USART_MspDeInit (USART_HandleTypeDef * huart)</code>
Function Description	DeInitialize the USART MSP.
Parameters	<ul style="list-style-type: none"> • <code>huart</code>: USART handle.
Return values	<ul style="list-style-type: none"> • None

44.2.9 HAL_USART_Transmit

Function Name	<code>HAL_StatusTypeDef HAL_USART_Transmit</code>
---------------	---

(USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)

Function Description	Simplex send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • husart: USART handle. • pTxData: Pointer to data buffer. • Size: Amount of data to be sent. • Timeout: Timeout duration.
Return values	<ul style="list-style-type: none"> • HAL status

44.2.10 HAL_USART_Receive

Function Name	HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • husart: USART handle. • pRxData: Pointer to data buffer. • Size: Amount of data to be received. • Timeout: Timeout duration.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • To receive synchronous data, dummy data are simultaneously transmitted.

44.2.11 HAL_USART_TransmitReceive

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Full-Duplex Send and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • husart: USART handle. • pTxData: pointer to TX data buffer. • pRxData: pointer to RX data buffer. • Size: amount of data to be sent (same amount to be received). • Timeout: Timeout duration.
Return values	<ul style="list-style-type: none"> • HAL status

44.2.12 HAL_USART_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • husart: USART handle. • pTxData: pointer to data buffer. • Size: amount of data to be sent.

- Return values
- HAL status

44.2.13 HAL_USART_Receive_IT

Function Name	HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • husart: USART handle. • pRxData: pointer to data buffer. • Size: amount of data to be received.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • To receive synchronous data, dummy data are simultaneously transmitted.

44.2.14 HAL_USART_TransmitReceive_IT

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Full-Duplex Send and Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • husart: USART handle. • pTxData: pointer to TX data buffer. • pRxData: pointer to RX data buffer. • Size: amount of data to be sent (same amount to be received).
Return values	<ul style="list-style-type: none"> • HAL status

44.2.15 HAL_USART_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • husart: USART handle. • pTxData: pointer to data buffer. • Size: amount of data to be sent.
Return values	<ul style="list-style-type: none"> • HAL status

44.2.16 HAL_USART_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • husart: USART handle. • pRxData: pointer to data buffer.

- **Size:** amount of data to be received.
- HAL status
- When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- The USART DMA transmit channel must be configured in order to generate the clock for the slave.

44.2.17 HAL_USART_TransmitReceive_DMA

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef *husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Full-Duplex Transmit Receive an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> • husart: USART handle. • pTxData: pointer to TX data buffer. • pRxData: pointer to RX data buffer. • Size: amount of data to be received/sent.
Return values	• HAL status
Notes	• When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

44.2.18 HAL_USART_DMAPause

Function Name	HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef *husart)
Function Description	Pause the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • husart: USART handle.
Return values	• HAL status

44.2.19 HAL_USART_DMAResume

Function Name	HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef *husart)
Function Description	Resume the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • husart: USART handle.
Return values	• HAL status

44.2.20 HAL_USART_DMAStop

Function Name	HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef *husart)
Function Description	Stop the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • husart: USART handle.
Return values	• HAL status

44.2.21 HAL_USART_IRQHandler

Function Name	void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)
Function Description	Handle USART interrupt request.
Parameters	<ul style="list-style-type: none">• husart: USART handle.
Return values	<ul style="list-style-type: none">• None

44.2.22 HAL_USART_TxCpltCallback

Function Name	void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none">• husart: USART handle.
Return values	<ul style="list-style-type: none">• None

44.2.23 HAL_USART_TxHalfCpltCallback

Function Name	void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none">• husart: USART handle
Return values	<ul style="list-style-type: none">• None

44.2.24 HAL_USART_RxCpltCallback

Function Name	void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)
Function Description	Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none">• husart: USART handle.
Return values	<ul style="list-style-type: none">• None

44.2.25 HAL_USART_RxHalfCpltCallback

Function Name	void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)
Function Description	Rx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none">• husart: USART handle.
Return values	<ul style="list-style-type: none">• None

44.2.26 HAL_USART_TxRxCpltCallback

Function Name	void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx/Rx Transfers completed callback for the non-blocking process.

Parameters	<ul style="list-style-type: none"> husart: USART handle
Return values	<ul style="list-style-type: none"> None

44.2.27 HAL_USART_ErrorCallback

Function Name	void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)
Function Description	USART error callback.
Parameters	<ul style="list-style-type: none"> husart: USART handle.
Return values	<ul style="list-style-type: none"> None

44.2.28 HAL_USART_GetState

Function Name	HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)
Function Description	Return the USART handle state.
Parameters	<ul style="list-style-type: none"> husart: : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.
Return values	<ul style="list-style-type: none"> USART handle state

44.2.29 HAL_USART_GetError

Function Name	uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)
Function Description	Return the USART error code.
Parameters	<ul style="list-style-type: none"> husart: : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.
Return values	<ul style="list-style-type: none"> USART handle Error Code

44.3 USART Firmware driver defines

44.3.1 USART

USART Clock

USART_CLOCK_DISABLE USART clock disable

USART_CLOCK_ENABLE USART clock enable

USART Clock Phase

USART_PHASE_1EDGE USART frame phase on first clock transition

USART_PHASE_2EDGE USART frame phase on second clock transition

USART Clock Polarity

USART_POLARITY_LOW USART Clock signal is steady Low

USART_POLARITY_HIGH USART Clock signal is steady High

USART Error

HAL_USART_ERROR_NONE	No error
HAL_USART_ERROR_PE	Parity error
HAL_USART_ERROR_NE	Noise error
HAL_USART_ERROR_FE	frame error
HAL_USART_ERROR_ORE	Overrun error
HAL_USART_ERROR_DMA	DMA transfer error

USART Exported Macros

<code>_HAL_USART_RESET_HANDLE_STA TE</code>	Description: <ul style="list-style-type: none"> Reset USART handle state. Parameters: <ul style="list-style-type: none"> <code>_HANDLE_</code>: USART handle. Return value: <ul style="list-style-type: none"> None
<code>_HAL_USART_GET_FLAG</code>	Description: <ul style="list-style-type: none"> Check whether the specified USART flag is set or not. Parameters: <ul style="list-style-type: none"> <code>_HANDLE_</code>: specifies the USART Handle <code>_FLAG_</code>: specifies the flag to check. This parameter can be one of the following values: <ul style="list-style-type: none"> <code>USART_FLAG_RXACK</code>: Receive enable acknowledge flag <code>USART_FLAG_TEACK</code>: Transmit enable acknowledge flag <code>USART_FLAG_BUSY</code>: Busy flag <code>USART_FLAG_CTS</code>: CTS Change flag <code>USART_FLAG_TXE</code>: Transmit data register empty flag <code>USART_FLAG_TC</code>: Transmission Complete flag <code>USART_FLAG_RXNE</code>: Receive data register not empty flag <code>USART_FLAG_IDLE</code>: Idle Line detection flag <code>USART_FLAG_ORE</code>: OverRun Error flag <code>USART_FLAG_NE</code>: Noise Error flag <code>USART_FLAG_FE</code>: Framing Error flag <code>USART_FLAG_PE</code>: Parity Error flag Return value:

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

[__HAL_USART_CLEAR_FLAG](#)**Description:**

- Clear the specified USART pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
 - USART_CLEAR_PEF
 - USART_CLEAR_FEF
 - USART_CLEAR_NEF
 - USART_CLEAR_OREF
 - USART_CLEAR_IDLEF
 - USART_CLEAR_TCF
 - USART_CLEAR_CTSF
 - USART_CLEAR_RTOF
 - USART_CLEAR_EOBF
 - USART_CLEAR_CMF
 - USART_CLEAR_WUF

Return value:

- None

[__HAL_USART_CLEAR_PEFLAG](#)**Description:**

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

[__HAL_USART_CLEAR_FEFLAG](#)**Description:**

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

[__HAL_USART_CLEAR_NEFLAG](#)**Description:**

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

`_HAL_USART_CLEAR_OREFLAG`

Description:

- Clear the USART ORE pending flag.

Parameters:

- `_HANDLE_`: specifies the USART Handle.

Return value:

- None

`_HAL_USART_CLEAR_IDLEFLAG`

Description:

- Clear the USART IDLE pending flag.

Parameters:

- `_HANDLE_`: specifies the USART Handle.

Return value:

- None

`_HAL_USART_ENABLE_IT`

Description:

- Enable 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`

Description:

- Disable 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](#)

- Check 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:

- Check whether the specified USART interrupt source is enabled 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_CLEAR_IT](#)**Description:**

- Clear 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_REQ`: Receive Data flush Request
 - `USART_TXDATA_FLUSH_REQ`: Transmit data flush Request

Return value:

- None

`_HAL_USART_ONE_BIT_SAMPLE_ENABLE`

Description:

- Enable the USART one bit sample method.

Parameters:

- `_HANDLE_`: specifies the USART Handle.

Return value:

- None

`_HAL_USART_ONE_BIT_SAMPLE_DISABLE`

Description:

- Disable the USART one bit sample method.

Parameters:

- `_HANDLE_`: specifies the USART Handle.

Return value:

- None

`_HAL_USART_ENABLE`

Description:

- Enable USART.

Parameters:

- `_HANDLE_`: specifies the USART Handle.

Return value:

- None

`_HAL_USART_DISABLE`

Description:

- Disable USART.

Parameters:

- `__HANDLE__`: specifies the USART Handle.

Return value:

- None

USART Flags

<code>USART_FLAG_RXACK</code>	USART receive enable acknowledge flag
<code>USART_FLAG_TEACK</code>	USART transmit enable acknowledge flag
<code>USART_FLAG_BUSY</code>	USART busy flag
<code>USART_FLAG_CTS</code>	USART clear to send flag
<code>USART_FLAG_CTSIF</code>	USART clear to send interrupt flag
<code>USART_FLAG_TXE</code>	USART transmit data register empty
<code>USART_FLAG_TC</code>	USART transmission complete
<code>USART_FLAG_RXNE</code>	USART read data register not empty
<code>USART_FLAG_IDLE</code>	USART idle flag
<code>USART_FLAG_ORE</code>	USART overrun error
<code>USART_FLAG_NE</code>	USART noise error
<code>USART_FLAG_FE</code>	USART frame error
<code>USART_FLAG_PE</code>	USART parity error

USART Interruption Flags Mask

<code>USART_IT_MASK</code>	USART interruptions flags mask
----------------------------	--------------------------------

USART Interrupts Definition

<code>USART_IT_PE</code>	USART parity error interruption
<code>USART_IT_TXE</code>	USART transmit data register empty interruption
<code>USART_IT_TC</code>	USART transmission complete interruption
<code>USART_IT_RXNE</code>	USART read data register not empty interruption
<code>USART_IT_IDLE</code>	USART idle interruption
<code>USART_IT_ERR</code>	USART error interruption
<code>USART_IT_ORE</code>	USART overrun error interruption
<code>USART_IT_NE</code>	USART noise error interruption
<code>USART_IT_FE</code>	USART frame error interruption

USART Interruption Clear Flags

<code>USART_CLEAR_PEF</code>	Parity Error Clear Flag
<code>USART_CLEAR_FEF</code>	Framing Error Clear Flag
<code>USART_CLEAR_NEF</code>	Noise detected Clear Flag
<code>USART_CLEAR_OREF</code>	OverRun Error Clear Flag
<code>USART_CLEAR_IDLEF</code>	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 frame last data bit clock pulse not output to SCLK pin

USART_LASTBIT_ENABLE USART frame last data bit clock pulse output to SCLK pin

USART Mode

USART_MODE_RX RX mode

USART_MODE_TX TX mode

USART_MODE_TX_RX RX and TX mode

USART Parity

USART_PARITY_NONE No parity

USART_PARITY_EVEN Even parity

USART_PARITY_ODD Odd parity

USARTE_x Request Parameters

USART_RXDATA_FLUSH_REQUEST Receive Data flush Request

USART_TXDATA_FLUSH_REQUEST Transmit data flush Request

USART Number of Stop Bits

USART_STOPBITS_1

USART_STOPBITS_2

USART_STOPBITS_1_5

45 HAL USART Extension Driver

45.1 USARTEx Firmware driver defines

45.1.1 USARTEx

USARTEx Exported Macros

<code>_HAL_USART_FLUSH_DRREGISTER</code>	Description:
	<ul style="list-style-type: none">Flush the USART Data registers.
	Parameters:
	<ul style="list-style-type: none"><code>_HANDLE_</code>: specifies the USART Handle.
	Return value:
	<ul style="list-style-type: none">None

USARTEx Word Length

<code>USART_WORDLENGTH_7B</code>	7-bit long USART frame
<code>USART_WORDLENGTH_8B</code>	8-bit long USART frame
<code>USART_WORDLENGTH_9B</code>	9-bit long USART frame

46 HAL WWDG Generic Driver

46.1 WWDG Firmware driver registers structures

46.1.1 WWDG_InitTypeDef

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

46.1.2 WWDG_HandleTypeDefDef

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

46.2 WWDG Firmware driver API description

46.2.1 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 informs when a WWDG reset has occurred (check available with `__HAL_RCC_GET_FLAG(RCC_FLAG_WWDGRST)`).
- 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) = $1000 * (T[5;0] + 1) / \text{WWDG clock}$ where T[5;0] are the lowest 6 bits of Counter.
- WWDG Counter refresh is allowed between the following limits :
 - min time (mS) = $1000 * (\text{Counter-Window}) / \text{WWDG clock}$
 - max time (mS) = $1000 * (\text{Counter-0x40}) / \text{WWDG clock}$
- Min-max timeout value @48 MHz(PCLK): ~85,3us / ~5,46 ms

46.2.2 How to use this driver

- Enable WWDG APB1 clock using `__HAL_RCC_WWDG_CLK_ENABLE()`.
- Set the WWDG prescaler, refresh window and counter value using `HAL_WWDG_Init()` function.
- Start the WWDG using `HAL_WWDG_Start()` function. When the WWDG is enabled the counter value should be configured to a value greater than 0x40 to prevent generating an immediate reset.
- Optionally you can enable the Early Wakeup Interrupt (EWI) which is generated when the counter reaches 0x40, and then start the WWDG using `HAL_WWDG_Start_IT()`. At EWI `HAL_WWDG_WakeupCallback()` is executed and user can add his own code by customization of function pointer `HAL_WWDG_WakeupCallback()`. Once enabled, EWI interrupt cannot be disabled except by a system reset.
- 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_ENABLE_IT`: Enable the WWDG early wakeup interrupt
- `__HAL_WWDG_GET_IT_SOURCE`: Check the selected WWDG's interrupt source
- `__HAL_WWDG_GET_FLAG`: Get the selected WWDG's flag status
- `__HAL_WWDG_CLEAR_FLAG`: Clear the WWDG's pending flags

46.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the WWDG according to the specified parameters in the WWDG_InitTypeDef and initialize the associated handle.
- Deinitialize the WWDG peripheral.
- Initialize the WWDG MSP.
- Deinitialize the WWDG MSP.

This section contains the following APIs:

- [***HAL_WWDG_Init\(\)***](#)
- [***HAL_WWDG_DeInit\(\)***](#)
- [***HAL_WWDG_MspInit\(\)***](#)
- [***HAL_WWDG_MspDeInit\(\)***](#)

46.2.4 IO operation functions

This section provides functions allowing to:

- Start the WWDG.
- Refresh the WWDG.
- Handle WWDG interrupt request and associated function callback.

This section contains the following APIs:

- [***HAL_WWDG_Start\(\)***](#)
- [***HAL_WWDG_Start_IT\(\)***](#)
- [***HAL_WWDG_Refresh\(\)***](#)
- [***HAL_WWDG_IRQHandler\(\)***](#)
- [***HAL_WWDG_WakeupCallback\(\)***](#)

46.2.5 Peripheral State functions

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

This section contains the following APIs:

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

46.2.6 HAL_WWDG_Init

Function Name	<i>HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwdg)</i>
Function Description	Initialize the WWDG according to the specified parameters in the WWDG_InitTypeDef and initialize the associated handle.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

46.2.7 HAL_WWDG_DeInit

Function Name	<i>HAL_StatusTypeDef HAL_WWDG_DeInit (WWDG_HandleTypeDef * hwdg)</i>
Function Description	Deinitialize the WWDG peripheral.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

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

46.2.8 HAL_WWDG_MspInit

Function Name	void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwdg)
Function Description	Initialize the WWWDG MSP.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWWDG_HandleTypeDef structure that contains the configuration information for the specified WWWDG module.
Return values	<ul style="list-style-type: none"> • None

46.2.9 HAL_WWDG_MspDeInit

Function Name	void HAL_WWDG_MspDeInit (WWDG_HandleTypeDef * hwdg)
Function Description	Deinitialize the WWWDG MSP.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWWDG_HandleTypeDef structure that contains the configuration information for the specified WWWDG module.
Return values	<ul style="list-style-type: none"> • None

46.2.10 HAL_WWDG_Start

Function Name	HAL_StatusTypeDef HAL_WWDG_Start (WWDG_HandleTypeDef * hwdg)
Function Description	Start the WWWDG.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWWDG_HandleTypeDef structure that contains the configuration information for the specified WWWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

46.2.11 HAL_WWDG_Start_IT

Function Name	HAL_StatusTypeDef HAL_WWDG_Start_IT (WWDG_HandleTypeDef * hwdg)
Function Description	Start the WWWDG with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWWDG_HandleTypeDef structure that contains the configuration information for the specified WWWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

46.2.12 HAL_WWDG_Refresh

Function Name	HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwdg, uint32_t Counter)
Function Description	Refresh the WWWDG.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWWDG_HandleTypeDef structure that

	contains the configuration information for the specified WWDG module.
• Counter: value of counter to put in WWDG counter	
Return values	• HAL status

46.2.13 HAL_WWDG_IRQHandler

Function Name	void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwdg)
Function Description	Handle WWDG interrupt request.
Parameters	<ul style="list-style-type: none"> hwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled when calling HAL_WWDG_Start_IT function. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

46.2.14 HAL_WWDG_WakeupCallback

Function Name	void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwdg)
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"> hwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> None

46.2.15 HAL_WWDG_GetState

Function Name	HAL_WWDG_StateTypeDef HAL_WWDG_GetState (WWDG_HandleTypeDef * hwdg)
Function Description	Return the WWDG handle state.
Parameters	<ul style="list-style-type: none"> hwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> HAL state

46.3 WWDG Firmware driver defines

46.3.1 WWDG

WWDG Exported Macros

<code>__HAL_WWDG_RESET_HANDLE_STATE</code>	Description: <ul style="list-style-type: none">• Reset WWDG handle state. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: WWDG handle Return value: <ul style="list-style-type: none">• None
<code>__HAL_WWDG_ENABLE</code>	Description: <ul style="list-style-type: none">• Enable the WWDG peripheral. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: WWDG handle Return value: <ul style="list-style-type: none">• None
<code>__HAL_WWDG_DISABLE</code>	Description: <ul style="list-style-type: none">• Disable the WWDG peripheral. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: WWDG handle Return value: <ul style="list-style-type: none">• None Notes: <ul style="list-style-type: none">• WARNING: This is a dummy macro for HAL code alignment. Once enable, WWDG Peripheral cannot be disabled except by a system reset.
<code>__HAL_WWDG_ENABLE_IT</code>	Description: <ul style="list-style-type: none">• Enable the WWDG early wakeup interrupt. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: WWDG handle• <code>__INTERRUPT__</code>: specifies the interrupt to enable. This parameter can be one of the following values:<ul style="list-style-type: none">– <code>WWDG_IT_EWI</code>: Early wakeup interrupt Return value: <ul style="list-style-type: none">• None Notes: <ul style="list-style-type: none">• Once enabled this interrupt cannot be disabled except by a system reset.
<code>__HAL_WWDG_DISABLE_IT</code>	Description:

- Disable the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to disable. This parameter can be one of the following values:
 - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment. Once enabled this interrupt cannot be disabled except by a system reset.

__HAL_WWDG_GET_IT

Description:

- Check whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the it to check. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

- The new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_IT

Description:

- Clear the WWDG interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: WWDG handle

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- The new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_FLAG**Description:**

- Clear the WWDG's pending flags.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- None

__HAL_WWDG_GET_IT_SOURCE**Description:**

- Check whether the specified WWDG interrupt source is enabled or not.

Parameters:

- __HANDLE__: WWDG Handle.
- __INTERRUPT__: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
 - WWDG_IT_EWI: Early Wakeup Interrupt

Return value:

- state: of __INTERRUPT__ (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1 WWDG counter clock = (PCLK1/4096)/1

WWDG_PRESCALER_2 WWDG counter clock = (PCLK1/4096)/2

WWDG_PRESCALER_4 WWDG counter clock = (PCLK1/4096)/4

WWDG_PRESCALER_8 WWDG counter clock = (PCLK1/4096)/8

47

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_MspDeInit() functions?

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

These functions are declared in 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.

48 Revision history

Table 26: Document revision history

Date	Revision	Changes
15-Oct-2014	1	Initial release.
09-Nov-2015	2	Macros and functions renamed for all APIs. Updated common macros in <i>HAL common resources</i> . Added LSE_STARTUP_TIMEOUT in <i>HAL configuration</i> .

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved