

Description of STM32L0xx HAL drivers

Introduction

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

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL0 for STM32L0 series)
 - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
 - A consistent set of middleware components such as RTOS, USB...
 - All embedded software utilities coming with a full set of examples.

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

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

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

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

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

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

This user manual is structured as follows:

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



Contents

1	Acronyms and definitions.....	37
2	Overview of HAL drivers	39
2.1	HAL and user-application files.....	39
2.1.1	HAL driver files	39
2.1.2	User-application files	40
2.2	HAL data structures	42
2.2.1	Peripheral handle structures	42
2.2.2	Initialization and configuration structure	44
2.2.3	Specific process structures	44
2.3	API classification	44
2.4	Devices supported by HAL drivers	45
2.5	HAL drivers rules.....	50
2.5.1	HAL API naming rules	50
2.5.2	HAL general naming rules	51
2.5.3	HAL interrupt handler and callback functions.....	52
2.6	HAL generic APIs.....	53
2.7	HAL extension APIs	54
2.7.1	HAL extension model overview	54
2.7.2	HAL extension model cases	55
2.8	File inclusion model.....	57
2.9	HAL common resources.....	58
2.10	HAL configuration.....	58
2.11	HAL system peripheral handling	59
2.11.1	Clock.....	59
2.11.2	GPIOs.....	60
2.11.3	Cortex NVIC and SysTick timer.....	62
2.11.4	PWR	62
2.11.5	EXTI.....	62
2.11.6	DMA.....	63
2.12	How to use HAL drivers	65
2.12.1	HAL usage models	65
2.12.2	HAL initialization	66
2.12.3	HAL IO operation process	69
2.12.4	Timeout and error management.....	72
3	HAL System Driver	76

3.1	HAL Firmware driver API description	76
3.1.1	How to use this driver	76
3.1.2	Initialization and de-initialization functions	76
3.1.3	HAL Control functions.....	76
3.1.4	HAL_Init.....	77
3.1.5	HAL_DeInit	77
3.1.6	HAL_MspInit	78
3.1.7	HAL_MspDeInit	78
3.1.8	HAL_InitTick	78
3.1.9	HAL_IncTick	78
3.1.10	HAL_GetTick	78
3.1.11	HAL_Delay	79
3.1.12	HAL_SuspendTick.....	79
3.1.13	HAL_ResumeTick.....	79
3.1.14	HAL_GetHalVersion	79
3.1.15	HAL_GetREVID	79
3.1.16	HAL_GetDEVID	80
3.1.17	HAL_DBGMCU_EnableDBGSleepMode	80
3.1.18	HAL_DBGMCU_DisableDBGSleepMode	80
3.1.19	HAL_DBGMCU_EnableDBGStopMode	80
3.1.20	HAL_DBGMCU_DisableDBGStopMode	80
3.1.21	HAL_DBGMCU_EnableDBGStandbyMode	80
3.1.22	HAL_DBGMCU_DisableDBGStandbyMode	80
3.1.23	HAL_DBGMCU_DBG_EnableLowPowerConfig	80
3.1.24	HAL_DBGMCU_DBG_DisableLowPowerConfig	81
3.1.25	HAL_SYSCFG_GetBootMode	81
3.1.26	HAL_SYSCFG_EnableVREFINT	81
3.1.27	HAL_SYSCFG_DisableVREFINT	81
3.1.28	HAL_SYSCFG_VREFINT_OutputSelect	82
3.1.29	HAL_SYSCFG_Enable_Lock_VREFINT	82
3.1.30	HAL_SYSCFG_Disable_Lock_VREFINT.....	82
3.2	HAL Firmware driver defines.....	82
3.2.1	HAL.....	82
4	HAL ADC Generic Driver.....	86
4.1	ADC Firmware driver registers structures	86
4.1.1	ADC_OversamplingTypeDef	86
4.1.2	ADC_InitTypeDef.....	86
4.1.3	ADC_HandleTypeDef	88

4.1.4	ADC_ChannelConfTypeDef	89
4.1.5	ADC_AnalogWDGConfTypeDef.....	89
4.2	ADC Firmware driver API description.....	90
4.2.1	ADC peripheral features	90
4.2.2	How to use this driver	90
4.2.3	IO operation functions	93
4.2.4	Peripheral Control functions	93
4.2.5	ADC Peripheral State functions.....	93
4.2.6	HAL_ADC_Init	93
4.2.7	HAL_ADC_DeInit.....	94
4.2.8	HAL_ADC_MspInit	94
4.2.9	HAL_ADC_MspDeInit.....	95
4.2.10	HAL_ADC_Start	95
4.2.11	HAL_ADC_Stop.....	95
4.2.12	HAL_ADC_PollForConversion	95
4.2.13	HAL_ADC_PollForEvent	96
4.2.14	HAL_ADC_Start_IT	96
4.2.15	HAL_ADC_Stop_IT	96
4.2.16	HAL_ADC_Start_DMA	96
4.2.17	HAL_ADC_Stop_DMA.....	96
4.2.18	HAL_ADC_GetValue	97
4.2.19	HAL_ADC_IRQHandler	97
4.2.20	HAL_ADC_ConvCpltCallback	97
4.2.21	HAL_ADC_ConvHalfCpltCallback.....	97
4.2.22	HAL_ADC_LevelOutOfWindowCallback	97
4.2.23	HAL_ADC_ErrorCallback	98
4.2.24	HAL_ADC_ConfigChannel	98
4.2.25	HAL_ADC_AnalogWDGConfig	98
4.2.26	HAL_ADC_GetState	99
4.2.27	HAL_ADC_GetError	99
4.3	ADC Firmware driver defines	99
4.3.1	ADC	99
5	HAL ADC Extension Driver	112
5.1	ADCEX Firmware driver API description	112
5.1.1	ADC specific features	112
5.1.2	How to use this driver	112
5.1.3	HAL_ADCEX_Calibration_Start.....	112
5.1.4	HAL_ADCEX_Calibration_GetValue	112

5.1.5	HAL_ADCEx_Calibration_SetValue	112
5.1.6	HAL_ADCEx_EnableVREFINT	113
5.1.7	HAL_ADCEx_DisableVREFINT	113
5.1.8	HAL_ADCEx_EnableVREFINTTempSensor	113
5.1.9	HAL_ADCEx_DisableVREFINTTempSensor	113
5.2	ADCEx Firmware driver defines	114
5.2.1	ADCEx	114
6	HAL COMP Generic Driver	115
6.1	COMP Firmware driver registers structures	115
6.1.1	COMP_InitTypeDef	115
6.1.2	COMP_HandleTypeDef	115
6.2	COMP Firmware driver API description	116
6.2.1	Initialization and de-initialization functions	116
6.2.2	IO operation functions	116
6.2.3	Peripheral Control functions	116
6.2.4	Peripheral State functions	116
6.2.5	HAL_COMP_Init	116
6.2.6	HAL_COMP_DeInit	117
6.2.7	HAL_COMP_MspInit	117
6.2.8	HAL_COMP_MspDeInit	117
6.2.9	HAL_COMP_Start	117
6.2.10	HAL_COMP_Stop	118
6.2.11	HAL_COMP_Start_IT	118
6.2.12	HAL_COMP_Stop_IT	118
6.2.13	HAL_COMP_IRQHandler	118
6.2.14	HAL_COMP_Lock	118
6.2.15	HAL_COMP_GetOutputLevel	118
6.2.16	HAL_COMP_TriggerCallback	119
6.2.17	HAL_COMP_GetState	119
6.3	COMP Firmware driver defines	119
6.3.1	COMP	119
7	HAL COMP Extension Driver	126
7.1	COMPEX Firmware driver API description	126
7.1.1	COMP peripheral Extended features	126
7.1.2	HAL_COMPEX_EnableVREFINT	126
7.1.3	HAL_COMPEX_DisableVREFINT	126
8	HAL CORTEX Generic Driver	127

8.1	CORTEX Firmware driver registers structures	127
8.1.1	MPU_Region_InitTypeDef.....	127
8.2	CORTEX Firmware driver API description	128
8.2.1	How to use this driver	128
8.2.2	Initialization and de-initialization functions	128
8.2.3	Peripheral Control functions	129
8.2.4	HAL_NVIC_SetPriority	129
8.2.5	HAL_NVIC_EnableIRQ	129
8.2.6	HAL_NVIC_DisableIRQ.....	130
8.2.7	HAL_NVIC_SystemReset.....	130
8.2.8	HAL_SYSTICK_Config.....	130
8.2.9	HAL_MPU_Disable.....	130
8.2.10	HAL_MPU_Enable	130
8.2.11	HAL_NVIC_GetPriority	131
8.2.12	HAL_NVIC_SetPendingIRQ.....	131
8.2.13	HAL_NVIC_GetPendingIRQ	131
8.2.14	HAL_NVIC_ClearPendingIRQ.....	131
8.2.15	HAL_SYSTICK_CLKSourceConfig	131
8.2.16	HAL_SYSTICK_IRQHandler	132
8.2.17	HAL_SYSTICK_Callback	132
8.2.18	HAL_MPU_ConfigRegion.....	132
8.3	CORTEX Firmware driver defines.....	132
8.3.1	CORTEX.....	132
9	HAL CRC Generic Driver.....	136
9.1	CRC Firmware driver registers structures	136
9.1.1	CRC_InitTypeDef	136
9.1.2	CRC_HandleTypeDef.....	137
9.2	CRC Firmware driver API description	137
9.2.1	Initialization and de-initialization functions	137
9.2.2	Peripheral Control functions	138
9.2.3	Peripheral State functions	138
9.2.4	HAL_CRC_Init	138
9.2.5	HAL_CRC_DeInit	138
9.2.6	HAL_CRC_MspInit	138
9.2.7	HAL_CRC_MspDeInit.....	138
9.2.8	HAL_CRC_Accumulate	139
9.2.9	HAL_CRC_Calculate.....	139
9.2.10	HAL_CRC_GetState.....	139

9.3	CRC Firmware driver defines	140
9.3.1	CRC	140
10	HAL CRC Extension Driver	142
10.1	CRCEX Firmware driver API description	142
10.1.1	CRC Extended features functions	142
10.1.2	HAL_CRCEX_Polynomial_Set	142
10.1.3	HAL_CRCEX_Input_Data_Reverse	142
10.1.4	HAL_CRCEX_Output_Data_Reverse.....	143
10.2	CRCEX Firmware driver defines	143
10.2.1	CRCEX.....	143
11	HAL CRYPT Generic Driver.....	145
11.1	CRYPT Firmware driver registers structures	145
11.1.1	CRYPT_InitTypeDef	145
11.1.2	CRYPT_HandleTypeDef.....	145
11.2	CRYPT Firmware driver API description	146
11.2.1	Initialization and de-initialization functions	146
11.2.2	AES processing functions	146
11.2.3	CRYPT IRQ handler management.....	147
11.2.4	Peripheral State functions	147
11.2.5	DMA callback functions	147
11.2.6	HAL_CRYPT_Init.....	147
11.2.7	HAL_CRYPT_DeInit	147
11.2.8	HAL_CRYPT_MspInit.....	148
11.2.9	HAL_CRYPT_MspDeInit	148
11.2.10	HAL_CRYPT_AESECB_Encrypt.....	148
11.2.11	HAL_CRYPT_AESCBC_Encrypt	148
11.2.12	HAL_CRYPT_AESCTR_Encrypt.....	149
11.2.13	HAL_CRYPT_AESECB_Decrypt	149
11.2.14	HAL_CRYPT_AESCBC_Decrypt	149
11.2.15	HAL_CRYPT_AESCTR_Decrypt	150
11.2.16	HAL_CRYPT_AESECB_Encrypt_IT	150
11.2.17	HAL_CRYPT_AESCBC_Encrypt_IT	150
11.2.18	HAL_CRYPT_AESCTR_Encrypt_IT	151
11.2.19	HAL_CRYPT_AESECB_Decrypt_IT	151
11.2.20	HAL_CRYPT_AESCBC_Decrypt_IT	151
11.2.21	HAL_CRYPT_AESCTR_Decrypt_IT	152
11.2.22	HAL_CRYPT_AESECB_Encrypt_DMA.....	152

11.2.23	HAL_Cryp_AESCBC_Encrypt_DMA	152
11.2.24	HAL_Cryp_AESCTR_Encrypt_DMA.....	153
11.2.25	HAL_Cryp_AESECB_Decrypt_DMA	153
11.2.26	HAL_Cryp_AESCBC_Decrypt_DMA	153
11.2.27	HAL_Cryp_AESCTR_Decrypt_DMA	153
11.2.28	HAL_Cryp_IRQHandler.....	154
11.2.29	HAL_Cryp_GetState	154
11.2.30	HAL_Cryp_ErrorCallback.....	154
11.2.31	HAL_Cryp_InCpltCallback	154
11.2.32	HAL_Cryp_OutCpltCallback	155
11.3	CRYP Firmware driver defines.....	155
11.3.1	CRYP.....	155
12	HAL CRYP Extension Driver	159
12.1	CRYPEx Firmware driver API description	159
12.1.1	Extended features functions	159
12.1.2	HAL_CRYPEx_ComputationCpltCallback.....	159
13	HAL DAC Generic Driver.....	160
13.1	DAC Firmware driver registers structures	160
13.1.1	DAC_HandleTypeDef	160
13.1.2	DAC_ChannelConfTypeDef	160
13.2	DAC Firmware driver API description.....	161
13.2.1	DAC Peripheral features.....	161
13.2.2	How to use this driver	162
13.2.3	Initialization and de-initialization functions	163
13.2.4	IO operation functions	163
13.2.5	Peripheral Control functions	164
13.2.6	Peripheral State and Errors functions	164
13.2.7	HAL_DAC_Init	164
13.2.8	HAL_DAC_DeInit.....	164
13.2.9	HAL_DAC_MspInit	164
13.2.10	HAL_DAC_MspDeInit.....	165
13.2.11	HAL_DAC_Start	165
13.2.12	HAL_DAC_Stop.....	165
13.2.13	HAL_DAC_Start_DMA	165
13.2.14	HAL_DAC_Stop_DMA.....	166
13.2.15	HAL_DAC_GetValue	166
13.2.16	HAL_DAC_IRQHandler	166
13.2.17	HAL_DAC_ConvCpltCallbackCh1.....	166

13.2.18	HAL_DAC_ConvHalfCpltCallbackCh1	167
13.2.19	HAL_DAC_ErrorCallbackCh1	167
13.2.20	HAL_DAC_DMAUnderrunCallbackCh1	167
13.2.21	HAL_DAC_SetValue	167
13.2.22	HAL_DAC_ConfigChannel	168
13.2.23	HAL_DAC_GetState	168
13.2.24	HAL_DAC_GetError	168
13.2.25	HAL_DAC_SetValue	169
13.3	DAC Firmware driver defines	169
13.3.1	DAC	169
14	HAL DAC Extension Driver	173
14.1	DACEx Firmware driver API description	173
14.1.1	How to use this driver	173
14.1.2	HAL_DACEx_DualGetValue	173
14.1.3	HAL_DACEx_TriangleWaveGenerate	173
14.1.4	HAL_DACEx_NoiseWaveGenerate	174
14.1.5	HAL_DACEx_DualSetValue	174
14.1.6	HAL_DACEx_ConvCpltCallbackCh2	175
14.1.7	HAL_DACEx_ConvHalfCpltCallbackCh2	175
14.1.8	HAL_DACEx_ErrorCallbackCh2	175
14.1.9	HAL_DACEx_DMAUnderrunCallbackCh2	175
14.1.10	HAL_DAC_Start	176
14.1.11	HAL_DAC_Start_DMA	176
14.1.12	HAL_DAC_Stop_DMA	176
14.1.13	HAL_DAC_GetValue	177
14.1.14	HAL_DAC_IRQHandler	177
14.1.15	HAL_DAC_SetValue	177
14.2	DACEx Firmware driver defines	178
14.2.1	DACEx	178
15	HAL DMA Generic Driver	180
15.1	DMA Firmware driver registers structures	180
15.1.1	DMA_InitTypeDef	180
15.1.2	DMA_HandleTypeDef	181
15.2	DMA Firmware driver API description	181
15.2.1	Initialization and de-initialization functions	181
15.2.2	IO operation functions	182
15.2.3	Peripheral State functions	182

15.2.4	HAL_DMA_Init.....	182
15.2.5	HAL_DMA_DeInit	182
15.2.6	HAL_DMA_Start	182
15.2.7	HAL_DMA_Start_IT	183
15.2.8	HAL_DMA_Abort	183
15.2.9	HAL_DMA_PollForTransfer.....	183
15.2.10	HAL_DMA_IRQHandler.....	184
15.2.11	HAL_DMA_GetState	184
15.2.12	HAL_DMA_GetError	184
15.3	DMA Firmware driver defines.....	184
15.3.1	DMA.....	184
16	HAL FIREWALL Generic Driver	191
16.1	FIREWALL Firmware driver registers structures	191
16.1.1	FIREWALL_InitTypeDef	191
16.2	FIREWALL Firmware driver API description	192
16.2.1	How to use this driver	192
16.2.2	Initialization and Configuration functions.....	192
16.2.3	HAL_FIREWALL_Config	192
16.2.4	HAL_FIREWALL_GetConfig	192
16.2.5	HAL_FIREWALL_EnableFirewall.....	193
16.2.6	HAL_FIREWALL_EnablePreArmFlag	193
16.2.7	HAL_FIREWALL_DisablePreArmFlag	193
16.3	FIREWALL Firmware driver defines.....	194
16.3.1	FIREWALL.....	194
17	HAL FLASH Generic Driver.....	199
17.1	FLASH Firmware driver registers structures	199
17.1.1	FLASH_EraseInitTypeDef	199
17.1.2	FLASH_ProcessTypeDef	199
17.2	FLASH Firmware driver API description.....	200
17.2.1	FLASH peripheral features	200
17.2.2	How to use this driver	200
17.2.3	Programming operation functions	201
17.2.4	Option Bytes Programming functions.....	201
17.2.5	Peripheral Control functions	202
17.2.6	Peripheral Errors functions	202
17.2.7	HAL_FLASH_Program	202
17.2.8	HAL_FLASH_Program_IT	202

17.2.9	HAL_FLASH_IRQHandler	203
17.2.10	HAL_FLASH_EndOfOperationCallback	203
17.2.11	HAL_FLASH_OperationErrorCallback	203
17.2.12	HAL_FLASH_Unlock	203
17.2.13	HAL_FLASH_Lock	203
17.2.14	HAL_FLASH_OB_Unlock	203
17.2.15	HAL_FLASH_OB_Lock	204
17.2.16	HAL_FLASH_OB_Launch	204
17.2.17	HAL_FLASH_GetError	204
17.3	FLASH Firmware driver defines	204
17.3.1	FLASH	204
18	HAL FLASH Extension Driver	208
18.1	FLASHEx Firmware driver registers structures	208
18.1.1	FLASH_OBProgramInitTypeDef	208
18.1.2	FLASH_AdvOBProgramInitTypeDef	209
18.2	FLASHEx Firmware driver API description.....	209
18.2.1	Flash peripheral Extended features	209
18.2.2	How to use this driver	209
18.2.3	FLASH Erasing Programming functions.....	210
18.2.4	Option Bytes Programming functions.....	210
18.2.5	DATA EEPROM Programming functions	211
18.2.6	HAL_FLASHEx_Erase	211
18.2.7	HAL_FLASHEx_Erase_IT	211
18.2.8	HAL_FLASHEx_OBProgram.....	212
18.2.9	HAL_FLASHEx_OBGetConfig	212
18.2.10	HAL_FLASHEx_AdvOBProgram	212
18.2.11	HAL_FLASHEx_AdvOBGetConfig	212
18.2.12	HAL_FLASHEx_OB_SelectPCROP	212
18.2.13	HAL_FLASHEx_OB_DeSelectPCROP	213
18.2.14	HAL_FLASHEx_DATAEEPROM_Unlock	213
18.2.15	HAL_FLASHEx_DATAEEPROM_Lock.....	213
18.2.16	HAL_FLASHEx_DATAEEPROM_Erase	213
18.2.17	HAL_FLASHEx_DATAEEPROM_Program.....	214
18.2.18	HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram	214
18.2.19	HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram.....	214
18.3	FLASHEx Firmware driver defines	214
18.3.1	FLASHEx	214

19	HAL FLASH__RAMFUNC Generic Driver	221
19.1	FLASH__RAMFUNC Firmware driver API description	221
19.1.1	HAL_FLASHEx_EnableRunPowerDown	221
19.1.2	HAL_FLASHEx_DisableRunPowerDown	221
19.1.3	HAL_FLASHEx_HalfPageProgram	221
19.1.4	HAL_FLASHRAM_GetError	222
19.1.5	HAL_FLASHEx_EraseParallelPage	222
19.1.6	HAL_FLASHEx_ProgramParallelHalfPage	222
20	HAL GPIO Generic Driver	224
20.1	GPIO Firmware driver registers structures	224
20.1.1	GPIO_InitTypeDef	224
20.2	GPIO Firmware driver API description	224
20.2.1	GPIO Peripheral features	224
20.2.2	How to use this driver	225
20.2.3	Initialization and de-initialization functions	225
20.2.4	IO operation functions	226
20.2.5	HAL_GPIO_Init	226
20.2.6	HAL_GPIO_DeInit	226
20.2.7	HAL_GPIO_ReadPin	226
20.2.8	HAL_GPIO_WritePin	227
20.2.9	HAL_GPIO_TogglePin	227
20.2.10	HAL_GPIO_LockPin	227
20.2.11	HAL_GPIO_EXTI_IRQHandler	228
20.2.12	HAL_GPIO_EXTI_Callback	228
20.3	GPIO Firmware driver defines	228
20.3.1	GPIO	228
21	HAL GPIO Extension Driver	232
21.1	GPIOEx Firmware driver defines	232
21.1.1	GPIOEx	232
22	HAL I2C Generic Driver	234
22.1	I2C Firmware driver registers structures	234
22.1.1	I2C_InitTypeDef	234
22.1.2	I2C_HandleTypeDef	234
22.2	I2C Firmware driver API description	235
22.2.1	How to use this driver	235
22.2.2	Initialization and de-initialization functions	238
22.2.3	IO operation functions	238

22.2.4	Peripheral State and Errors functions	240
22.2.5	HAL_I2C_Init	240
22.2.6	HAL_I2C_DeInit.....	240
22.2.7	HAL_I2C_MspInit	240
22.2.8	HAL_I2C_MspDeInit.....	240
22.2.9	HAL_I2C_Master_Transmit.....	241
22.2.10	HAL_I2C_Master_Receive	241
22.2.11	HAL_I2C_Slave_Transmit.....	241
22.2.12	HAL_I2C_Slave_Receive	241
22.2.13	HAL_I2C_Master_Transmit_IT.....	242
22.2.14	HAL_I2C_Master_Receive_IT.....	242
22.2.15	HAL_I2C_Slave_Transmit_IT.....	242
22.2.16	HAL_I2C_Slave_Receive_IT.....	242
22.2.17	HAL_I2C_Master_Transmit_DMA	243
22.2.18	HAL_I2C_Master_Receive_DMA.....	243
22.2.19	HAL_I2C_Slave_Transmit_DMA.....	243
22.2.20	HAL_I2C_Slave_Receive_DMA.....	244
22.2.21	HAL_I2C_Mem_Write.....	244
22.2.22	HAL_I2C_Mem_Read	244
22.2.23	HAL_I2C_Mem_Write_IT	245
22.2.24	HAL_I2C_Mem_Read_IT	245
22.2.25	HAL_I2C_Mem_Write_DMA	245
22.2.26	HAL_I2C_Mem_Read_DMA	246
22.2.27	HAL_I2C_IsDeviceReady.....	246
22.2.28	HAL_I2C_EV_IRQHandler	246
22.2.29	HAL_I2C_ER_IRQHandler	246
22.2.30	HAL_I2C_MasterTxCpltCallback.....	247
22.2.31	HAL_I2C_MasterRxCpltCallback	247
22.2.32	HAL_I2C_SlaveTxCpltCallback.....	247
22.2.33	HAL_I2C_SlaveRxCpltCallback	247
22.2.34	HAL_I2C_MemTxCpltCallback.....	247
22.2.35	HAL_I2C_MemRxCpltCallback	248
22.2.36	HAL_I2C_ErrorCallback	248
22.2.37	HAL_I2C_GetState	248
22.2.38	HAL_I2C_GetError	248
22.3	I2C Firmware driver defines	248
22.3.1	I2C	248
23	HAL I2C Extension Driver	254

23.1	I2CEx Firmware driver API description	254
23.1.1	I2C peripheral Extended features.....	254
23.1.2	How to use this driver	254
23.1.3	Extended features functions	254
23.1.4	HAL_I2CEx_ConfigAnalogFilter	254
23.1.5	HAL_I2CEx_ConfigDigitalFilter	254
23.1.6	HAL_I2CEx_EnableWakeUp.....	255
23.1.7	HAL_I2CEx_DisableWakeUp.....	255
23.1.8	HAL_I2CEx_EnableFastModePlus	255
23.1.9	HAL_I2CEx_DisableFastModePlus.....	256
23.2	I2CEx Firmware driver defines	256
23.2.1	I2CEx.....	256
24	HAL I2S Generic Driver	257
24.1	I2S Firmware driver registers structures	257
24.1.1	I2S_InitTypeDef.....	257
24.1.2	I2S_HandleTypeDef	257
24.2	I2S Firmware driver API description.....	258
24.2.1	How to use this driver	258
24.2.2	Initialization and de-initialization functions	260
24.2.3	IO operation functions	260
24.2.4	Peripheral State and Errors functions	261
24.2.5	HAL_I2S_Init	261
24.2.6	HAL_I2S_DeInit.....	261
24.2.7	HAL_I2S_MspInit.....	261
24.2.8	HAL_I2S_MspDeInit.....	262
24.2.9	HAL_I2S_Transmit	262
24.2.10	HAL_I2S_Receive	262
24.2.11	HAL_I2S_Transmit_IT	263
24.2.12	HAL_I2S_Receive_IT	263
24.2.13	HAL_I2S_Transmit_DMA	264
24.2.14	HAL_I2S_Receive_DMA	264
24.2.15	HAL_I2S_DMAPause	265
24.2.16	HAL_I2S_DMAResume.....	265
24.2.17	HAL_I2S_DMAStop.....	265
24.2.18	HAL_I2S_IRQHandler	265
24.2.19	HAL_I2S_TxHalfCpltCallback	265
24.2.20	HAL_I2S_TxCpltCallback	266
24.2.21	HAL_I2S_RxHalfCpltCallback	266

24.2.22	HAL_I2S_RxCpltCallback	266
24.2.23	HAL_I2S_ErrorCallback	266
24.2.24	HAL_I2S_GetState	266
24.2.25	HAL_I2S_GetError	266
24.3	I2S Firmware driver defines	267
24.3.1	I2S	267
25	HAL IRDA Generic Driver.....	271
25.1	IRDA Firmware driver registers structures	271
25.1.1	IRDA_InitTypeDef.....	271
25.1.2	IRDA_HandleTypeDef.....	271
25.2	IRDA Firmware driver API description.....	272
25.2.1	Initialization and Configuration functions.....	272
25.2.2	IO operation functions	273
25.2.3	Peripheral Control functions	274
25.2.4	HAL_IRDA_Init	274
25.2.5	HAL_IRDA_DeInit.....	274
25.2.6	HAL_IRDA_MspInit	274
25.2.7	HAL_IRDA_MspDeInit.....	274
25.2.8	HAL_IRDA_Transmit	275
25.2.9	HAL_IRDA_Receive	275
25.2.10	HAL_IRDA_Transmit_IT	275
25.2.11	HAL_IRDA_Receive_IT	275
25.2.12	HAL_IRDA_Transmit_DMA.....	276
25.2.13	HAL_IRDA_Receive_DMA.....	276
25.2.14	HAL_IRDA_DMAPause.....	276
25.2.15	HAL_IRDA_DMAResume.....	276
25.2.16	HAL_IRDA_DMAStop.....	276
25.2.17	HAL_IRDA_IRQHandler	277
25.2.18	HAL_IRDA_TxHalfCpltCallback	277
25.2.19	HAL_IRDA_TxCpltCallback.....	277
25.2.20	HAL_IRDA_RxHalfCpltCallback.....	277
25.2.21	HAL_IRDA_RxCpltCallback	277
25.2.22	HAL_IRDA_ErrorCallback	278
25.2.23	HAL_IRDA_GetState.....	278
25.2.24	HAL_IRDA_GetError	278
25.3	IRDA Firmware driver defines	278
25.3.1	IRDA	278

26	HAL IRDA Extension Driver	288
26.1	IRDAEx Firmware driver defines	288
26.1.1	IRDAEx.....	288
27	HAL IWDG Generic Driver.....	289
27.1	IWDG Firmware driver registers structures	289
27.1.1	IWDG_InitTypeDef	289
27.1.2	IWDG_HandleTypeDef.....	289
27.2	IWDG Firmware driver API description	290
27.2.1	IWDG Generic features	290
27.2.2	How to use this driver	290
27.2.3	Initialization and de-initialization functions	291
27.2.4	IO operation functions	291
27.2.5	Peripheral State functions	291
27.2.6	HAL_IWDG_Init.....	291
27.2.7	HAL_IWDG_MspInit	291
27.2.8	HAL_IWDG_Start	292
27.2.9	HAL_IWDG_Refresh	292
27.2.10	HAL_IWDG_GetState.....	292
27.3	IWDG Firmware driver defines	292
27.3.1	IWDG.....	292
28	HAL LCD Generic Driver	295
28.1	LCD Firmware driver registers structures.....	295
28.1.1	LCD_InitTypeDef	295
28.1.2	LCD_HandleTypeDef	296
28.2	LCD Firmware driver API description	296
28.2.1	How to use this driver	296
28.2.2	Initialization and Configuration functions.....	297
28.2.3	IO operation functions	297
28.2.4	Peripheral State functions	297
28.2.5	HAL_LCD_DeInit	297
28.2.6	HAL_LCD_Init.....	298
28.2.7	HAL_LCD_MspDeInit	298
28.2.8	HAL_LCD_MspInit.....	298
28.2.9	HAL_LCD_Write	298
28.2.10	HAL_LCD_Clear	299
28.2.11	HAL_LCD_UpdateDisplayRequest	299
28.2.12	HAL_LCD_GetState	299

28.2.13	HAL_LCD_GetError.....	300
28.3	LCD Firmware driver defines.....	300
28.3.1	LCD.....	300
29	HAL LPTIM Generic Driver.....	310
29.1	LPTIM Firmware driver registers structures	310
29.1.1	LPTIM_ClockConfigTypeDef.....	310
29.1.2	LPTIM_ULPClockConfigTypeDef.....	310
29.1.3	LPTIM_TriggerConfigTypeDef	310
29.1.4	LPTIM_InitTypeDef.....	311
29.1.5	LPTIM_HandleTypeDef.....	311
29.2	LPTIM Firmware driver API description.....	312
29.2.1	Initialization and de-initialization functions	312
29.2.2	LPTIM Start Stop operation functions	312
29.2.3	LPTIM Read operation functions	313
29.2.4	LPTIM IRQ handler.....	313
29.2.5	Peripheral State functions	313
29.2.6	HAL_LPTIM_Init	314
29.2.7	HAL_LPTIM_DeInit.....	314
29.2.8	HAL_LPTIM_MspInit	314
29.2.9	HAL_LPTIM_MspDeInit.....	314
29.2.10	HAL_LPTIM_PWM_Start.....	314
29.2.11	HAL_LPTIM_PWM_Stop.....	315
29.2.12	HAL_LPTIM_PWM_Start_IT	315
29.2.13	HAL_LPTIM_PWM_Stop_IT	315
29.2.14	HAL_LPTIM_OnePulse_Start	315
29.2.15	HAL_LPTIM_OnePulse_Stop.....	315
29.2.16	HAL_LPTIM_OnePulse_Start_IT	316
29.2.17	HAL_LPTIM_OnePulse_Stop_IT	316
29.2.18	HAL_LPTIM_SetOnce_Start	316
29.2.19	HAL_LPTIM_SetOnce_Stop	316
29.2.20	HAL_LPTIM_SetOnce_Start_IT	316
29.2.21	HAL_LPTIM_SetOnce_Stop_IT	317
29.2.22	HAL_LPTIM_Encoder_Start.....	317
29.2.23	HAL_LPTIM_Encoder_Stop	317
29.2.24	HAL_LPTIM_Encoder_Start_IT.....	317
29.2.25	HAL_LPTIM_Encoder_Stop_IT	318
29.2.26	HAL_LPTIM_TimeOut_Start	318
29.2.27	HAL_LPTIM_TimeOut_Stop.....	318

29.2.28	HAL_LPTIM_TimeOut_Start_IT	318
29.2.29	HAL_LPTIM_TimeOut_Stop_IT	318
29.2.30	HAL_LPTIM_Counter_Start	319
29.2.31	HAL_LPTIM_Counter_Stop.....	319
29.2.32	HAL_LPTIM_Counter_Start_IT	319
29.2.33	HAL_LPTIM_Counter_Stop_IT	319
29.2.34	HAL_LPTIM_ReadCounter	319
29.2.35	HAL_LPTIM_ReadAutoReload	320
29.2.36	HAL_LPTIM_ReadCompare	320
29.2.37	HAL_LPTIM_IRQHandler	320
29.2.38	HAL_LPTIM_CompareMatchCallback	320
29.2.39	HAL_LPTIM_AutoReloadMatchCallback	320
29.2.40	HAL_LPTIM_TriggerCallback.....	320
29.2.41	HAL_LPTIM_CompareWriteCallback.....	321
29.2.42	HAL_LPTIM_AutoReloadWriteCallback.....	321
29.2.43	HAL_LPTIM_DirectionUpCallback	321
29.2.44	HAL_LPTIM_DirectionDownCallback.....	321
29.2.45	HAL_LPTIM_GetState.....	321
29.3	LPTIM Firmware driver defines	322
29.3.1	LPTIM	322
30	HAL LPTIM Extension Driver	328
30.1	LPTIMEx Firmware driver defines	328
30.1.1	LPTIMEx	328
31	HAL PCD Generic Driver	329
31.1	PCD Firmware driver registers structures	329
31.1.1	PCD_InitTypeDef.....	329
31.1.2	PCD_EPTTypeDef.....	329
31.1.3	PCD_HandleTypeDef	330
31.2	PCD Firmware driver API description.....	331
31.2.1	How to use this driver	331
31.2.2	Initialization and de-initialization functions	332
31.2.3	IO operation functions	332
31.2.4	Peripheral Control functions	332
31.2.5	Peripheral State functions	332
31.2.6	HAL_PCD_Init	333
31.2.7	HAL_PCD_DeInit.....	333
31.2.8	HAL_PCD_MspInit	333
31.2.9	HAL_PCD_MspDeInit.....	333

31.2.10	HAL_PCD_Start	333
31.2.11	HAL_PCD_Stop.....	333
31.2.12	HAL_PCD_IRQHandler	334
31.2.13	HAL_PCD_DataOutStageCallback	334
31.2.14	HAL_PCD_DataInStageCallback	334
31.2.15	HAL_PCD_SetupStageCallback	334
31.2.16	HAL_PCD_SOFCallback.....	334
31.2.17	HAL_PCD_ResetCallback.....	334
31.2.18	HAL_PCD_SuspendCallback	335
31.2.19	HAL_PCD_ResumeCallback.....	335
31.2.20	HAL_PCD_ISOOUTIncompleteCallback.....	335
31.2.21	HAL_PCD_ISOINIncompleteCallback.....	335
31.2.22	HAL_PCD_ConnectCallback.....	335
31.2.23	HAL_PCD_DisconnectCallback	336
31.2.24	HAL_PCD_DevConnect	336
31.2.25	HAL_PCD_DevDisconnect.....	336
31.2.26	HAL_PCD_SetAddress	336
31.2.27	HAL_PCD_EP_Open	336
31.2.28	HAL_PCD_EP_Close	337
31.2.29	HAL_PCD_EP_Receive	337
31.2.30	HAL_PCD_EP_GetRxCount	337
31.2.31	HAL_PCD_EP_Transmit	337
31.2.32	HAL_PCD_EP_SetStall.....	337
31.2.33	HAL_PCD_EP_ClrStall.....	338
31.2.34	HAL_PCD_EP_Flush	338
31.2.35	HAL_PCD_ActivateRemoteWakeup	338
31.2.36	HAL_PCD_DeActivateRemoteWakeup.....	338
31.2.37	PCD_WritePMA.....	338
31.2.38	PCD_ReadPMA.....	339
31.2.39	HAL_PCD_GetState.....	339
31.3	PCD Firmware driver defines	339
31.3.1	PCD	339
32	HAL PCD Extension Driver	348
32.1	PCDEx Firmware driver API description	348
32.1.1	Peripheral extended features functions.....	348
32.1.2	HAL_PCDEx_PMAConfig	348
33	HAL PWR Generic Driver	349

33.1	PWR Firmware driver registers structures	349
33.1.1	PWR_PVDTypeDef	349
33.2	PWR Firmware driver API description	349
33.2.1	Initialization and de-initialization functions	349
33.2.2	Peripheral Control functions	349
33.2.3	HAL_PWR_DeInit	353
33.2.4	HAL_PWR_EnableBkUpAccess	353
33.2.5	HAL_PWR_DisableBkUpAccess	353
33.2.6	HAL_PWR_EnableBkUpAccess	354
33.2.7	HAL_PWR_DisableBkUpAccess	354
33.2.8	HAL_PWR_ConfigPVD	354
33.2.9	HAL_PWR_EnablePVD	354
33.2.10	HAL_PWR_DisablePVD	354
33.2.11	HAL_PWR_EnableWakeUpPin	355
33.2.12	HAL_PWR_DisableWakeUpPin	355
33.2.13	HAL_PWR_EnterSLEEPMode	355
33.2.14	HAL_PWR_EnterSTOPMode	355
33.2.15	HAL_PWR_EnterSTANDBYMode	356
33.2.16	HAL_PWR_EnableSleepOnExit	356
33.2.17	HAL_PWR_DisableSleepOnExit	356
33.2.18	HAL_PWR_EnableSEVOnPend	357
33.2.19	HAL_PWR_DisableSEVOnPend	357
33.2.20	HAL_PWR_PVD_IRQHandler	357
33.2.21	HAL_PWR_PVDCallback	357
33.3	PWR Firmware driver defines	357
33.3.1	PWR	357
34	HAL PWR Extension Driver	363
34.1	PWREx Firmware driver defines	363
34.1.1	PWREx	363
35	HAL RCC Generic Driver	364
35.1	RCC Firmware driver registers structures	364
35.1.1	RCC_PLLInitTypeDef	364
35.1.2	RCC_OscInitTypeDef	364
35.1.3	RCC_ClkInitTypeDef	365
35.2	RCC Firmware driver API description	366
35.2.1	RCC specific features	366
35.2.2	RCC Limitations	366

35.2.3	Initialization and de-initialization functions	366
35.2.4	Peripheral Control functions	367
35.2.5	HAL_RCC_DeInit	368
35.2.6	HAL_RCC_OscConfig	368
35.2.7	HAL_RCC_ClockConfig	368
35.2.8	HAL_RCC_MCOConfig	369
35.2.9	HAL_RCC_EnableCSS	369
35.2.10	HAL_RCC_GetSysClockFreq	370
35.2.11	HAL_RCC_GetHCLKFreq	370
35.2.12	HAL_RCC_GetPCLK1Freq	371
35.2.13	HAL_RCC_GetPCLK2Freq	371
35.2.14	HAL_RCC_GetOscConfig	371
35.2.15	HAL_RCC_GetClockConfig	371
35.2.16	HAL_RCC_NMI_IRQHandler	371
35.2.17	HAL_RCC_CSSCallback	372
35.3	RCC Firmware driver defines	372
35.3.1	RCC	372
36	HAL RCC Extension Driver	392
36.1	RCCEX Firmware driver registers structures	392
36.1.1	RCC_PeriphCLKInitTypeDef	392
36.1.2	RCC_CRSSInitTypeDef	393
36.1.3	RCC_CRSSynchroInfoTypeDef	393
36.2	RCCEX Firmware driver API description	394
36.2.1	Extended Peripheral Control functions	394
36.2.2	HAL_RCC_DeInit	394
36.2.3	HAL_RCCEX_PeriphCLKConfig	395
36.2.4	HAL_RCCEX_GetPeriphCLKConfig	395
36.2.5	HAL_RCCEX_GetPeriphCLKFreq	395
36.2.6	HAL_RCCEX_EnableLSECSS	395
36.2.7	HAL_RCCEX_DisableLSECSS	396
36.2.8	HAL_RCCEX_EnableLSECSS_IT	396
36.2.9	HAL_RCCEX_LSECSS_IRQHandler	396
36.2.10	HAL_RCCEX_LSECSS_Callback	396
36.2.11	HAL_RCCEX_CRSSConfig	396
36.2.12	HAL_RCCEX_CRSSoftwareSynchronizationGenerate	396
36.2.13	HAL_RCCEX_CRSSGetSynchronizationInfo	396
36.2.14	HAL_RCCEX_CRSSWaitSynchronization	397
36.2.15	HAL_RCCEX_EnableHSI48_VREFINT	397

36.2.16	HAL_RCCEX_DisableHSI48_VREFINT	397
36.3	RCCEX Firmware driver defines	397
36.3.1	RCCEX	397
37	HAL RNG Generic Driver	420
37.1	RNG Firmware driver registers structures	420
37.1.1	RNG_HandleTypeDef	420
37.2	RNG Firmware driver API description	420
37.2.1	How to use this driver	420
37.2.2	Initialization and de-initialization functions	420
37.2.3	Peripheral Control functions	421
37.2.4	Peripheral State functions	421
37.2.5	HAL_RNG_Init	421
37.2.6	HAL_RNG_DeInit	421
37.2.7	HAL_RNG_MspInit	421
37.2.8	HAL_RNG_MspDeInit	422
37.2.9	HAL_RNG_GenerateRandomNumber	422
37.2.10	HAL_RNG_GenerateRandomNumber_IT	422
37.2.11	HAL_RNG_IRQHandler	422
37.2.12	HAL_RNG_GetRandomNumber	423
37.2.13	HAL_RNG_GetRandomNumber_IT	423
37.2.14	HAL_RNG_ReadLastRandomNumber	423
37.2.15	HAL_RNG_ReadyDataCallback	423
37.2.16	HAL_RNG_ErrorCallback	423
37.2.17	HAL_RNG_GetState	423
37.3	RNG Firmware driver defines	424
37.3.1	RNG	424
38	HAL RTC Generic Driver	427
38.1	RTC Firmware driver registers structures	427
38.1.1	RTC_InitTypeDef	427
38.1.2	RTC_TimeTypeDef	427
38.1.3	RTC_DateTypeDef	428
38.1.4	RTC_AlarmTypeDef	429
38.1.5	RTC_HandleTypeDef	429
38.2	RTC Firmware driver API description	430
38.2.1	Backup Domain Operating Condition	430
38.2.2	Backup Domain Reset	430
38.2.3	Backup Domain Access	430

38.2.4	How to use RTC Driver.....	430
38.2.5	RTC and low power modes	431
38.2.6	Initialization and de-initialization functions	431
38.2.7	RTC Time and Date functions	432
38.2.8	RTC Alarm functions	432
38.2.9	Peripheral Control functions	432
38.2.10	Peripheral State functions	432
38.2.11	HAL_RTC_Init	432
38.2.12	HAL_RTC_DeInit.....	432
38.2.13	HAL_RTC_MspInit.....	433
38.2.14	HAL_RTC_MspDeInit.....	433
38.2.15	HAL_RTC_SetTime.....	433
38.2.16	HAL_RTC_GetTime	433
38.2.17	HAL_RTC_SetDate	434
38.2.18	HAL_RTC_GetDate.....	434
38.2.19	HAL_RTC_SetAlarm	434
38.2.20	HAL_RTC_SetAlarm_IT	435
38.2.21	HAL_RTC_DeactivateAlarm.....	435
38.2.22	HAL_RTC_GetAlarm.....	435
38.2.23	HAL_RTC_AlarmIRQHandler.....	436
38.2.24	HAL_RTC_AlarmAEventCallback	436
38.2.25	HAL_RTC_PollForAlarmAEvent.....	436
38.2.26	HAL_RTC_WaitForSynchro	436
38.2.27	HAL_RTC_GetState	437
38.3	RTC Firmware driver defines	437
38.3.1	RTC	437
39	HAL RTC Extension Driver	447
39.1	RTCEX Firmware driver registers structures	447
39.1.1	RTC_TamperTypeDef	447
39.2	RTCEX Firmware driver API description.....	448
39.2.1	RTC TimeStamp and Tamper functions.....	448
39.2.2	RTC Wake-up functions	448
39.2.3	Extended Peripheral Control functions.....	448
39.2.4	Extended features functions.....	449
39.2.5	HAL_RTCEX_SetTimeStamp	449
39.2.6	HAL_RTCEX_SetTimeStamp_IT	449
39.2.7	HAL_RTCEX_DeactivateTimeStamp	450
39.2.8	HAL_RTCEX_GetTimeStamp.....	450

39.2.9	HAL_RTCEX_SetTamper	450
39.2.10	HAL_RTCEX_SetTamper_IT	451
39.2.11	HAL_RTCEX_DeactivateTamper	451
39.2.12	HAL_RTCEX_TamperTimeStampIRQHandler	451
39.2.13	HAL_RTCEX_TimeStampEventCallback	451
39.2.14	HAL_RTCEX_Tamper1EventCallback	452
39.2.15	HAL_RTCEX_Tamper2EventCallback	452
39.2.16	HAL_RTCEX_Tamper3EventCallback	452
39.2.17	HAL_RTCEX_PollForTimeStampEvent	452
39.2.18	HAL_RTCEX_PollForTamper1Event	452
39.2.19	HAL_RTCEX_PollForTamper2Event	452
39.2.20	HAL_RTCEX_PollForTamper3Event	453
39.2.21	HAL_RTCEX_SetWakeUpTimer	453
39.2.22	HAL_RTCEX_SetWakeUpTimer_IT	453
39.2.23	HAL_RTCEX_DeactivateWakeUpTimer	453
39.2.24	HAL_RTCEX_GetWakeUpTimer	453
39.2.25	HAL_RTCEX_WakeUpTimerIRQHandler	454
39.2.26	HAL_RTCEX_WakeUpTimerEventCallback	454
39.2.27	HAL_RTCEX_PollForWakeUpTimerEvent	454
39.2.28	HAL_RTCEX_BKUPWrite	454
39.2.29	HAL_RTCEX_BKUPRead	454
39.2.30	HAL_RTCEX_SetSmoothCalib	455
39.2.31	HAL_RTCEX_SetSynchroShift	455
39.2.32	HAL_RTCEX_SetCalibrationOutPut	456
39.2.33	HAL_RTCEX_DeactivateCalibrationOutPut	456
39.2.34	HAL_RTCEX_SetRefClock	456
39.2.35	HAL_RTCEX_DeactivateRefClock	456
39.2.36	HAL_RTCEX_EnableBypassShadow	457
39.2.37	HAL_RTCEX_DisableBypassShadow	457
39.2.38	HAL_RTCEX_AlarmBEEventCallback	457
39.2.39	HAL_RTCEX_PollForAlarmBEEvent	457
39.3	RTCEX Firmware driver defines	457
39.3.1	RTCEX	457
40	HAL SMARTCARD Generic Driver.....	479
40.1	SMARTCARD Firmware driver registers structures	479
40.1.1	SMARTCARD_InitTypeDef	479
40.1.2	SMARTCARD_AdvFeatureInitTypeDef	480
40.1.3	SMARTCARD_HandleTypeDef	481

40.2	SMARTCARD Firmware driver API description.....	482
40.2.1	Initialization and Configuration functions.....	482
40.2.2	IO operation functions	482
40.2.3	Peripheral State functions	483
40.2.4	HAL_SMARTCARD_Init.....	483
40.2.5	HAL_SMARTCARD_DeInit	483
40.2.6	HAL_SMARTCARD_MspInit	483
40.2.7	HAL_SMARTCARD_MspDeInit	483
40.2.8	HAL_SMARTCARD_Transmit.....	484
40.2.9	HAL_SMARTCARD_Receive.....	484
40.2.10	HAL_SMARTCARD_Transmit_IT	484
40.2.11	HAL_SMARTCARD_Receive_IT	484
40.2.12	HAL_SMARTCARD_Transmit_DMA.....	485
40.2.13	HAL_SMARTCARD_Receive_DMA.....	485
40.2.14	HAL_SMARTCARD_IRQHandler.....	485
40.2.15	HAL_SMARTCARD_TxCpltCallback	485
40.2.16	HAL_SMARTCARD_RxCpltCallback	485
40.2.17	HAL_SMARTCARD_ErrorCallback.....	486
40.2.18	HAL_SMARTCARD_GetState	486
40.2.19	HAL_SMARTCARD_GetError.....	486
40.3	SMARTCARD Firmware driver defines	486
40.3.1	SMARTCARD.....	486
41	HAL SMARTCARD Extension Driver	499
41.1	SMARTCARDEx Firmware driver API description	499
41.1.1	How to use this driver	499
41.1.2	Peripheral Control functions	499
41.1.3	HAL_SMARTCARDEx_BlockLength_Config	499
41.1.4	HAL_SMARTCARDEx_TimeOut_Config	499
41.1.5	HAL_SMARTCARDEx_EnableReceiverTimeOut	500
41.1.6	HAL_SMARTCARDEx_DisableReceiverTimeOut	500
41.2	SMARTCARDEx Firmware driver defines.....	500
41.2.1	SMARTCARDEx.....	500
42	HAL SMBUS Generic Driver.....	501
42.1	SMBUS Firmware driver registers structures	501
42.1.1	SMBUS_InitTypeDef	501
42.1.2	SMBUS_HandleTypeDef.....	502
42.2	SMBUS Firmware driver API description	503

42.2.1	Initialization and de-initialization functions	503
42.2.2	IO operation functions	503
42.2.3	Peripheral State and Errors functions	504
42.2.4	HAL_SMBUS_Init	504
42.2.5	HAL_SMBUS_DeInit	504
42.2.6	HAL_SMBUS_MspInit	505
42.2.7	HAL_SMBUS_MspDeInit	505
42.2.8	HAL_SMBUS_Master_Transmit_IT	505
42.2.9	HAL_SMBUS_Master_Receive_IT	505
42.2.10	HAL_SMBUS_Master_Abort_IT	506
42.2.11	HAL_SMBUS_Slave_Transmit_IT	506
42.2.12	HAL_SMBUS_Slave_Receive_IT	506
42.2.13	HAL_SMBUS_EnableListen_IT	507
42.2.14	HAL_SMBUS_DisableListen_IT	507
42.2.15	HAL_SMBUS_EnableAlert_IT	507
42.2.16	HAL_SMBUS_DisableAlert_IT	507
42.2.17	HAL_SMBUS_IsDeviceReady	508
42.2.18	HAL_SMBUS_EV_IRQHandler	508
42.2.19	HAL_SMBUS_ER_IRQHandler	508
42.2.20	HAL_SMBUS_MasterTxCpltCallback	508
42.2.21	HAL_SMBUS_MasterRxCpltCallback	508
42.2.22	HAL_SMBUS_SlaveTxCpltCallback	509
42.2.23	HAL_SMBUS_SlaveRxCpltCallback	509
42.2.24	HAL_SMBUS_AddrCallback	509
42.2.25	HAL_SMBUS_ListenCpltCallback	509
42.2.26	HAL_SMBUS_ErrorCallback	510
42.2.27	HAL_SMBUS_GetState	510
42.2.28	HAL_SMBUS_GetError	510
42.3	SMBUS Firmware driver defines	510
42.3.1	SMBUS	510
43	HAL SPI Generic Driver	518
43.1	SPI Firmware driver registers structures	518
43.1.1	SPI_InitTypeDef	518
43.1.2	SPI_HandleTypeDef	519
43.2	SPI Firmware driver API description	520
43.2.1	How to use this driver	520
43.2.2	Initialization and de-initialization functions	521
43.2.3	IO operation functions	522

43.2.4	Peripheral State and Errors functions	523
43.2.5	HAL_SPI_Init	523
43.2.6	HAL_SPI_DeInit	523
43.2.7	HAL_SPI_MspInit	523
43.2.8	HAL_SPI_MspDeInit.....	524
43.2.9	HAL_SPI_Transmit.....	524
43.2.10	HAL_SPI_Receive	524
43.2.11	HAL_SPI_TransmitReceive.....	524
43.2.12	HAL_SPI_Transmit_IT.....	525
43.2.13	HAL_SPI_Receive_IT.....	525
43.2.14	HAL_SPI_TransmitReceive_IT	525
43.2.15	HAL_SPI_Transmit_DMA.....	525
43.2.16	HAL_SPI_Receive_DMA.....	526
43.2.17	HAL_SPI_TransmitReceive_DMA.....	526
43.2.18	HAL_SPI_DMAPause.....	526
43.2.19	HAL_SPI_DMAResume	526
43.2.20	HAL_SPI_DMAStop	527
43.2.21	HAL_SPI_IRQHandler.....	527
43.2.22	HAL_SPI_TxCpltCallback	527
43.2.23	HAL_SPI_RxCpltCallback	527
43.2.24	HAL_SPI_TxRxCpltCallback	527
43.2.25	HAL_SPI_TxHalfCpltCallback	528
43.2.26	HAL_SPI_RxHalfCpltCallback.....	528
43.2.27	HAL_SPI_TxRxHalfCpltCallback.....	528
43.2.28	HAL_SPI_ErrorCallback.....	528
43.2.29	HAL_SPI_GetState.....	528
43.2.30	HAL_SPI_GetError	528
43.3	SPI Firmware driver defines.....	529
43.3.1	SPI	529
44	HAL TIM Generic Driver	534
44.1	TIM Firmware driver registers structures.....	534
44.1.1	TIM_Base_InitTypeDef.....	534
44.1.2	TIM_OC_InitTypeDef.....	534
44.1.3	TIM_OnePulse_InitTypeDef	535
44.1.4	TIM_IC_InitTypeDef	535
44.1.5	TIM_Encoder_InitTypeDef	536
44.1.6	TIM_ClockConfigTypeDef	537
44.1.7	TIM_ClearInputConfigTypeDef.....	537

44.1.8	TIM_SlaveConfigTypeDef	538
44.1.9	TIM_HandleTypeDef	538
44.2	TIM Firmware driver API description	539
44.2.1	TIMER Generic features.....	539
44.2.2	How to use this driver	539
44.2.3	Timer Base functions.....	540
44.2.4	Peripheral State functions	540
44.2.5	HAL_TIM_Base_Init	540
44.2.6	HAL_TIM_Base_DeInit.....	541
44.2.7	HAL_TIM_Base_MspInit.....	541
44.2.8	HAL_TIM_Base_MspDeInit.....	541
44.2.9	HAL_TIM_Base_Start.....	541
44.2.10	HAL_TIM_Base_Stop.....	541
44.2.11	HAL_TIM_Base_Start_IT	542
44.2.12	HAL_TIM_Base_Stop_IT.....	542
44.2.13	HAL_TIM_Base_Start_DMA	542
44.2.14	HAL_TIM_Base_Stop_DMA.....	542
44.2.15	HAL_TIM_OC_Init	542
44.2.16	HAL_TIM_OC_DeInit.....	543
44.2.17	HAL_TIM_OC_MspInit	543
44.2.18	HAL_TIM_OC_MspDeInit.....	543
44.2.19	HAL_TIM_OC_Start	543
44.2.20	HAL_TIM_OC_Stop.....	543
44.2.21	HAL_TIM_OC_Start_IT	544
44.2.22	HAL_TIM_OC_Stop_IT	544
44.2.23	HAL_TIM_OC_Start_DMA	544
44.2.24	HAL_TIM_OC_Stop_DMA	544
44.2.25	HAL_TIM_PWM_Init.....	545
44.2.26	HAL_TIM_PWM_DeInit	545
44.2.27	HAL_TIM_PWM_MspInit.....	545
44.2.28	HAL_TIM_PWM_MspDeInit	545
44.2.29	HAL_TIM_PWM_Start.....	545
44.2.30	HAL_TIM_PWM_Stop	546
44.2.31	HAL_TIM_PWM_Start_IT	546
44.2.32	HAL_TIM_PWM_Stop_IT	546
44.2.33	HAL_TIM_PWM_Start_DMA.....	547
44.2.34	HAL_TIM_PWM_Stop_DMA	547
44.2.35	HAL_TIM_IC_Init	547
44.2.36	HAL_TIM_IC_DeInit	547

44.2.37	HAL_TIM_IC_MspInit	548
44.2.38	HAL_TIM_IC_MspDeInit.....	548
44.2.39	HAL_TIM_IC_Start	548
44.2.40	HAL_TIM_IC_Stop	548
44.2.41	HAL_TIM_IC_Start_IT	548
44.2.42	HAL_TIM_IC_Stop_IT	549
44.2.43	HAL_TIM_IC_Start_DMA	549
44.2.44	HAL_TIM_IC_Stop_DMA	549
44.2.45	HAL_TIM_OnePulse_Init.....	550
44.2.46	HAL_TIM_OnePulse_DeInit	550
44.2.47	HAL_TIM_OnePulse_MspInit.....	550
44.2.48	HAL_TIM_OnePulse_MspDeInit	550
44.2.49	HAL_TIM_OnePulse_Start	550
44.2.50	HAL_TIM_OnePulse_Stop	551
44.2.51	HAL_TIM_OnePulse_Start_IT.....	551
44.2.52	HAL_TIM_OnePulse_Stop_IT	551
44.2.53	HAL_TIM_Encoder_Init	551
44.2.54	HAL_TIM_Encoder_DeInit	552
44.2.55	HAL_TIM_Encoder_MspInit	552
44.2.56	HAL_TIM_Encoder_MspDeInit.....	552
44.2.57	HAL_TIM_Encoder_Start	552
44.2.58	HAL_TIM_Encoder_Stop	552
44.2.59	HAL_TIM_Encoder_Start_IT	553
44.2.60	HAL_TIM_Encoder_Stop_IT	553
44.2.61	HAL_TIM_Encoder_Start_DMA	553
44.2.62	HAL_TIM_Encoder_Stop_DMA	554
44.2.63	HAL_TIM_IRQHandler	554
44.2.64	HAL_TIM_OC_ConfigChannel	554
44.2.65	HAL_TIM_IC_ConfigChannel.....	554
44.2.66	HAL_TIM_PWM_ConfigChannel.....	555
44.2.67	HAL_TIM_OnePulse_ConfigChannel.....	555
44.2.68	HAL_TIM_DMABurst_WriteStart.....	555
44.2.69	HAL_TIM_DMABurst_WriteStop	556
44.2.70	HAL_TIM_DMABurst_ReadStart.....	556
44.2.71	HAL_TIM_DMABurst_ReadStop	557
44.2.72	HAL_TIM_GenerateEvent	557
44.2.73	HAL_TIM_ConfigOCrefClear	558
44.2.74	HAL_TIM_ConfigClockSource	558

44.2.75	HAL_TIM_ConfigTI1Input.....	558
44.2.76	HAL_TIM_SlaveConfigSynchronization	559
44.2.77	HAL_TIM_SlaveConfigSynchronization_IT	559
44.2.78	HAL_TIM_ReadCapturedValue.....	559
44.2.79	HAL_TIM_PeriodElapsedCallback	560
44.2.80	HAL_TIM_OC_DelayElapsedCallback.....	560
44.2.81	HAL_TIM_IC_CaptureCallback	560
44.2.82	HAL_TIM_PWM_PulseFinishedCallback	560
44.2.83	HAL_TIM_TriggerCallback	560
44.2.84	HAL_TIM_ErrorCallback.....	560
44.2.85	HAL_TIM_Base_GetState	561
44.2.86	HAL_TIM_OC_GetState.....	561
44.2.87	HAL_TIM_PWM_GetState	561
44.2.88	HAL_TIM_IC_GetState.....	561
44.2.89	HAL_TIM_OnePulse_GetState	561
44.2.90	HAL_TIM_Encoder_GetState.....	561
44.2.91	TIM_DMAError	562
44.2.92	TIM_DMADelayPulseCplt.....	562
44.2.93	TIM_DMACaptureCplt	562
44.3	TIM Firmware driver defines.....	562
44.3.1	TIM.....	562
45	HAL TIM Extension Driver.....	576
45.1	TIMEx Firmware driver registers structures.....	576
45.1.1	TIM_MasterConfigTypeDef	576
45.2	TIMEx Firmware driver API description	576
45.2.1	TIM specific features integration	576
45.2.2	How to use this driver	576
45.2.3	Peripheral Control functions	577
45.2.4	HAL_TIMEx_MasterConfigSynchronization	577
45.2.5	HAL_TIMEx_RemapConfig	577
45.3	TIMEx Firmware driver defines	578
45.3.1	TIMEx	578
46	HAL TSC Generic Driver	580
46.1	TSC Firmware driver registers structures.....	580
46.1.1	TSC_InitTypeDef	580
46.1.2	TSC_IOConfigTypeDef.....	581
46.1.3	TSC_HandleTypeDef	581

46.2	TSC Firmware driver API description	581
46.2.1	TSC specific features	581
46.2.2	How to use this driver	582
46.2.3	IO Operation functions.....	582
46.2.4	Peripheral Control functions	583
46.2.5	State functions.....	583
46.2.6	Initialization and de-initialization functions	583
46.2.7	HAL_TSC_Init.....	583
46.2.8	HAL_TSC_DeInit	584
46.2.9	HAL_TSC_MspInit.....	584
46.2.10	HAL_TSC_MspDeInit	584
46.2.11	HAL_TSC_Start.....	584
46.2.12	HAL_TSC_Start_IT.....	584
46.2.13	HAL_TSC_Stop	585
46.2.14	HAL_TSC_Stop_IT	585
46.2.15	HAL_TSC_GroupGetStatus	585
46.2.16	HAL_TSC_GroupGetValue	585
46.2.17	HAL_TSC_IOConfig	585
46.2.18	HAL_TSC_IODischarge	586
46.2.19	HAL_TSC_GetState	586
46.2.20	HAL_TSC_PollForAcquisition	586
46.2.21	HAL_TSC_IRQHandler	586
46.2.22	HAL_TSC_ConvCpltCallback.....	586
46.2.23	HAL_TSC_ErrorCallback.....	587
46.2.24	HAL_TSC_Init.....	587
46.2.25	HAL_TSC_DeInit	587
46.2.26	HAL_TSC_MspInit.....	587
46.2.27	HAL_TSC_MspDeInit	587
46.3	TSC Firmware driver defines.....	588
46.3.1	TSC.....	588
47	HAL UART Generic Driver.....	597
47.1	UART Firmware driver registers structures	597
47.1.1	UART_InitTypeDef	597
47.1.2	UART_AdvFeatureInitTypeDef.....	598
47.1.3	UART_HandleTypeDef.....	598
47.2	UART Firmware driver API description	599
47.2.1	Initialization and Configuration functions.....	599
47.2.2	IO operation functions	600

47.2.3	Peripheral Control functions	601
47.2.4	HAL_UART_Init	601
47.2.5	HAL_HalfDuplex_Init	601
47.2.6	HAL_LIN_Init	601
47.2.7	HAL_MultiProcessor_Init	602
47.2.8	HAL_UART_DeInit	602
47.2.9	HAL_UART_MspInit	602
47.2.10	HAL_UART_MspDeInit	603
47.2.11	HAL_UART_Transmit	603
47.2.12	HAL_UART_Receive	603
47.2.13	HAL_UART_Transmit_IT	603
47.2.14	HAL_UART_Receive_IT	603
47.2.15	HAL_UART_Transmit_DMA	604
47.2.16	HAL_UART_Receive_DMA	604
47.2.17	HAL_UART_DMAPause	604
47.2.18	HAL_UART_DMAResume	604
47.2.19	HAL_UART_DMAStop	604
47.2.20	HAL_UART_IRQHandler	605
47.2.21	HAL_UART_TxCpltCallback	605
47.2.22	HAL_UART_TxHalfCpltCallback	605
47.2.23	HAL_UART_RxCpltCallback	605
47.2.24	HAL_UART_RxHalfCpltCallback	605
47.2.25	HAL_UART_ErrorCallback	606
47.2.26	HAL_MultiProcessor_EnableMuteMode	606
47.2.27	HAL_MultiProcessor_DisableMuteMode	606
47.2.28	HAL_MultiProcessor_EnterMuteMode	606
47.2.29	HAL_HalfDuplex_EnableTransmitter	606
47.2.30	HAL_HalfDuplex_EnableReceiver	607
47.2.31	HAL_LIN_SendBreak	607
47.2.32	HAL_UART_GetState	607
47.2.33	HAL_UART_GetError	607
47.3	UART Firmware driver defines	607
47.3.1	UART	607
48	HAL UART Extension Driver	625
48.1	UARTEEx Firmware driver registers structures	625
48.1.1	UART_WakeUpTypeDef	625
48.2	UARTEEx Firmware driver API description	625
48.2.1	Initialization and Configuration functions	625

48.2.2	Peripheral Control funtions	625
48.2.3	HAL_RS485Ex_Init.....	626
48.2.4	HAL_UARTEx_EnableStopMode	626
48.2.5	HAL_UARTEx_EnableClockStopMode	626
48.2.6	HAL_UARTEx_DisableStopMode	627
48.2.7	HAL_UARTEx_DisableClockStopMode	627
48.2.8	HAL_UARTEx_StopModeWakeUpSourceConfig	627
48.2.9	HAL_MultiProcessorEx_AddressLength_Set.....	627
48.2.10	HAL_UARTEx_WakeupCallback	627
48.3	UARTEx Firmware driver defines.....	628
48.3.1	UARTEx.....	628
49	HAL USART Generic Driver	630
49.1	USART Firmware driver registers structures.....	630
49.1.1	USART_InitTypeDef	630
49.1.2	USART_HandleTypeDef	631
49.2	USART Firmware driver API description	632
49.2.1	Initialization and Configuration functions.....	632
49.2.2	IO operation functions	632
49.2.3	Peripheral State functions	633
49.2.4	HAL_USART_Init.....	634
49.2.5	HAL_USART_DeInit	634
49.2.6	HAL_USART_MspInit.....	634
49.2.7	HAL_USART_MspDeInit	634
49.2.8	HAL_USART_Transmit	634
49.2.9	HAL_USART_Receive	635
49.2.10	HAL_USART_TransmitReceive	635
49.2.11	HAL_USART_Transmit_IT	635
49.2.12	HAL_USART_Receive_IT	635
49.2.13	HAL_USART_TransmitReceive_IT	636
49.2.14	HAL_USART_Transmit_DMA	636
49.2.15	HAL_USART_Receive_DMA	636
49.2.16	HAL_USART_TransmitReceive_DMA	637
49.2.17	HAL_USART_DMAPause	637
49.2.18	HAL_USART_DMAResume	637
49.2.19	HAL_USART_DMAStop	637
49.2.20	HAL_USART_IRQHandler	637
49.2.21	HAL_USART_TxCpltCallback	638
49.2.22	HAL_USART_TxHalfCpltCallback.....	638

49.2.23	HAL_USART_RxCpltCallback.....	638
49.2.24	HAL_USART_RxHalfCpltCallback	638
49.2.25	HAL_USART_TxRxCpltCallback.....	638
49.2.26	HAL_USART_ErrorCallback	638
49.2.27	HAL_USART_GetState	639
49.2.28	HAL_USART_GetError.....	639
49.3	USART Firmware driver defines.....	639
49.3.1	USART.....	639
50	HAL USART Extension Driver	649
50.1	USARTEx Firmware driver defines	649
50.1.1	USARTEx	649
51	HAL WWDG Generic Driver	650
51.1	WWDG Firmware driver registers structures.....	650
51.1.1	WWDG_InitTypeDef	650
51.1.2	WWDG_HandleTypeDef	650
51.2	WWDG Firmware driver API description	651
51.2.1	WWDG specific features	651
51.2.2	How to use this driver	651
51.2.3	Initialization and de-initialization functions	651
51.2.4	IO operation functions	652
51.2.5	Peripheral State functions	652
51.2.6	HAL_WWDG_Init.....	652
51.2.7	HAL_WWDG_DeInit.....	652
51.2.8	HAL_WWDG_MspInit.....	652
51.2.9	HAL_WWDG_MspDeInit	653
51.2.10	HAL_WWDG_WakeupCallback	653
51.2.11	HAL_WWDG_Start.....	653
51.2.12	HAL_WWDG_Start_IT.....	653
51.2.13	HAL_WWDG_Refresh.....	653
51.2.14	HAL_WWDG_IRQHandler	654
51.2.15	HAL_WWDG_WakeupCallback	654
51.2.16	HAL_WWDG_GetState	654
51.3	WWDG Firmware driver defines.....	655
51.3.1	WWDG.....	655
52	FAQs.....	659
53	Revision history	663

List of tables

Table 1: Acronyms and definitions.....	37
Table 2: HAL drivers files.....	39
Table 3: User-application files	40
Table 4: APIs classification	45
Table 5: List of devices supported by HAL drivers	46
Table 6: HAL API naming rules	50
Table 7: Macros handling interrupts and specific clock configurations	51
Table 8: Callback functions.....	52
Table 9: HAL generic APIs	53
Table 10: HAL extension APIs	54
Table 11: Define statements used for HAL configuration	58
Table 12: Description of GPIO_InitTypeDef structure	60
Table 13: Description of EXTI configuration macros	63
Table 14: MSP functions.....	68
Table 15: Timeout values	72
Table 16: IRDA frame formats	272
Table 17: Number of wait states (WS) vs CPU clock (HCLK) frequency	367
Table 18: USART frame formats	482
Table 19: Maximum SPI frequency vs data size (8 bits)	521
Table 20: Maximum SPI frequency vs data size (16 bits)	521
Table 21: UART frame formats (2 M bits).....	600
Table 22: USART frame formats	632
Table 23: Document revision history	663

List of figures

Figure 1: Example of project template	42
Figure 2: Adding device-specific functions	55
Figure 3: Adding family-specific functions	55
Figure 4: Adding new peripherals	56
Figure 5: Updating existing APIs	56
Figure 6: File inclusion model	57
Figure 7: HAL driver model	65

1 Acronyms and definitions

Table 1: Acronyms and definitions

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

Acronym	Definition
TSC	Touch Sensing Controller
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

2 Overview of HAL drivers

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

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

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

The HAL main features are the following:

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

2.1 HAL and user-application files

2.1.1 HAL driver files

HAL drivers are composed of the following set of files:

Table 2: HAL drivers files

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

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

2.1.2 User-application files

The minimum files required to build an application using the HAL are listed in the table below:

Table 3: User-application files

File	Description
<i>system_stm32l0xx.c</i>	This file contains SystemInit() which is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows to relocate the vector table in internal SRAM.
<i>startup_stm32l0xx.s</i>	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
<i>stm32l0xx_flash.icf</i> (optional)	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
<i>stm32l0xx_flash.ld</i> (optional)	Linker file for SW4STM32 toolchain.
<i>stm32l0xx_hal_msp.c</i>	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<i>stm32l0xx_hal_conf.h</i>	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.

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

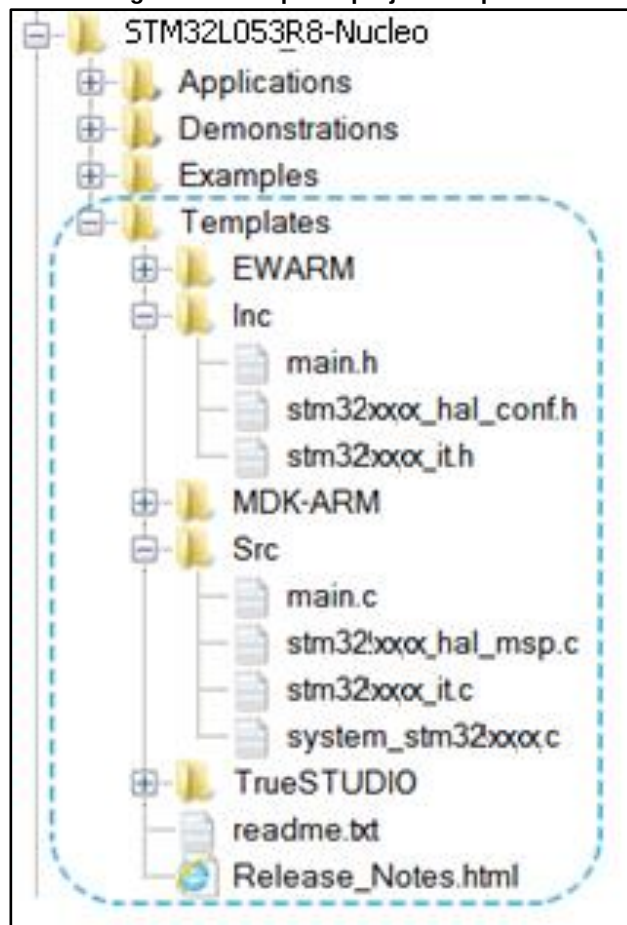
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Their characteristics are the following:

- 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 uint32_t ErrorCode; /* USART Error code */
}USART_HandleTypeDef;
```



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

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



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



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

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

2.2.2 Initialization and configuration structure

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

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



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

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

2.2.3 Specific process structures

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

Example:

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

2.3 API classification

The HAL APIs are classified into three categories:

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

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

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

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

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


```
#if !defined(STM32L051xx) && !defined(STM32L061xx) void
HAL_RCCEx_CRSSConfig(RCC_CRSSInitTypeDef *pInit); void
HAL_RCCEx_CRSSoftwareSynchronizationGenerate(void); RCC_CRSSStatusTypeDef
HAL_RCCEx_CRSSWaitSynchronization(uint32_t Timeout); #endif /* !(STM32L051xx) &&
!(STM32L061xx) */ The data structure related to the specific APIs is delimited by
the device part number define statement. It is located in the corresponding
extension header C file.
```

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

Table 4: APIs classification

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

Notes:

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



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



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

2.4 Devices supported by HAL drivers

Table 5: List of devices supported by HAL drivers

IP/Module	STM32L011xx	STM32L021xx	STM32L0"1xx	STM32L041xx	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx	STM32L071xx	STM32L072xx	STM32L073xx	STM32L081xx	STM32L082xx	STM32L083xx
stm32l0xx_hal_c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_adc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_comp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_cortex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_crc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_cryp.c	No	Yes	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
stm32l0xx_hal_cryp_ex.c	No	Yes	No	Yes	No	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
stm32l0xx_hal_dac.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_dac_ex.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_firewall.c	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_flash.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32L011xx	STM32L021xx	STM32L0"1xx	STM32L041xx	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx	STM32L071xx	STM32L072xx	STM32L073xx	STM32L081xx	STM32L082xx	STM32L083xx
stm32l0xx_hal_flash_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_flash_ramfunc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_i2c_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_i2s.c	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_lcd.c	No	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes	No	No	Yes
stm32l0xx_hal_lptim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_pcd.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_pcd_ex.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Overview of HAL drivers

UM1749

IP/Module	STM32L011xx	STM32L021xx	STM32L0"1xx	STM32L041xx	STM32L051xx	STM32L052xx	STM32L053xx	STM32L061xx	STM32L062xx	STM32L063xx	STM32L071xx	STM32L072xx	STM32L073xx	STM32L081xx	STM32L082xx	STM32L083xx
stm32l0xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rng.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_smbus.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal-smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal-smartcard_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal-tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal-tim_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal-tsc.c	No	No	No	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes
stm32l0xx_hal-uart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal-uart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32L 011xx	STM32L 021xx	STM32L 0"1xx	STM32L 041xx	STM32L 051xx	STM32L 052xx	STM32L 053xx	STM32L 061xx	STM32L 062xx	STM32L 063xx	STM32L 071xx	STM32L 072xx	STM32L 073xx	STM32L 081xx	STM32L 082xx	STM32L 083xx
stm32l0xx_hal_Usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l0xx_hal_Wwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

2.5 HAL drivers rules

2.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 6: HAL API naming rules

	Generic	Family specific	Device specific
File names	<i>stm32l0xx_hal_ppp (c/h)</i>	<i>stm32l0xx_hal_ppp_ex (c/h)</i>	<i>stm32l0xx_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 STM32L0xx reference manuals.
- Peripheral registers are declared in the *PPP_TypeDef* structure (e.g. *ADC_TypeDef*) in *stm32l0xxx.h* header file. *stm32l0xxx.h* corresponds to *stm32l051xx.h*, *stm32l052xx.h*, *stm32l053xx.h*, *stm32l061xx.h*, *stm32l062xx.h* or *stm32l063xx.h*.
- Peripheral function names are prefixed by *HAL_*, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (e.g. *HAL_UART_Transmit()*). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named *PPP_InitTypeDef* (e.g. *ADC_InitTypeDef*).
- The structure containing the Specific configuration parameters for the PPP peripheral are named *PPP_xxxxConfTypeDef* (e.g. *ADC_ChannelConfTypeDef*).
- Peripheral handle structures are named *PPP_HandleTypeDef* (e.g. *DMA_HandleTypeDef*).
- The functions used to initialize the PPP peripheral according to parameters specified in *PPP_InitTypeDef* are named *HAL_PPP_Init* (e.g. *HAL_TIM_Init()*).
- The functions used to reset the PPP peripheral registers to their default values are named *PPP_DeInit*, e.g. *TIM_DeInit*.

- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL_PPP_Function_DMA ()*.
- The **Feature** prefix should refer to the new feature.
Example: *HAL_ADC_Start()* refers to the injection mode

2.5.2 HAL general naming rules

- For the shared and system peripherals, no handle or instance object is used. This rule applies to the following peripherals:
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 */
}
```

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

Table 7: Macros handling interrupts and specific clock configurations

Macros	Description
<code>__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Enables a specific peripheral interrupt
<code>__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Disables a specific peripheral interrupt
<code>__HAL_PPP_GET_IT (__HANDLE__, __INTERRUPT __)</code>	Gets a specific peripheral interrupt status
<code>__HAL_PPP_CLEAR_IT (__HANDLE__, __INTERRUPT __)</code>	Clears a specific peripheral interrupt status
<code>__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG __)</code>	Gets a specific peripheral flag status
<code>__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG __)</code>	Clears a specific peripheral flag status
<code>__HAL_PPP_ENABLE(__HANDLE__)</code>	Enables a peripheral
<code>__HAL_PPP_DISABLE(__HANDLE__)</code>	Disables a peripheral
<code>__HAL_PPP_XXXX (__HANDLE__, __PARAM __)</code>	Specific PPP HAL driver macro
<code>__HAL_PPP_GET_IT_SOURCE (__HANDLE__, __INTERRUPT __)</code>	Checks the source of specified interrupt

- NVIC and SYSTICK are two ARM Cortex core features. The APIs related to these features are located in the *stm32l0xx_hal_cortex.c* file.
- When a status bit or a flag is read from registers, it is composed of shifted values depending on the number of read values and of their size. In this case, the returned

status width is 32 bits. Example: STATUS = XX | (YY << 16) or STATUS = XX | (YY << 8) | (YY << 16) | (YY << 24)".

- The PPP handles are valid before using the HAL_PPP_Init() API. The init function performs a check before modifying the handle fields.

```
HAL_PPP_Init(PPP_HandleTypeDef)

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

- The macros defined below are used:
 - Conditional macro:

```
#define ABS(x) (((x) > 0) ? (x) : -(x))
```

- Pseudo-code macro (multiple instructions macro):

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

2.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32l0xx_it.c
- User callback functions.

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

There are three types of user callbacks functions:

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

Table 8: Callback functions

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

2.6 HAL generic APIs

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

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

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

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

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

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

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

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

Table 9: HAL generic APIs

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

Function Group	Common API Name	Description
	<i>HAL_ADC_ConvCpltCallback()</i>	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	<i>HAL_ADC_ErrorCallback()</i>	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	<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
State and Errors group	<i>HAL_ADC_GetState()</i>	This function allows getting in runtime the peripheral and the data flow states.
	<i>HAL_ADC_GetError()</i>	This function allows getting in runtime the error that occurred during IT routine

2.7 HAL extension APIs

2.7.1 HAL extension model overview

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

The extension model consists of an additional file, *stm32l0xx_hal_ppp_ex.c*, that includes all the specific functions and define statements (*stm32l0xx_hal_ppp_ex.h*) for a given part number.

Below an example based on the ADC peripheral:

Table 10: HAL extension APIs

Function Group	Common API Name
<i>HAL_ADCEx_Calibration_Start()</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

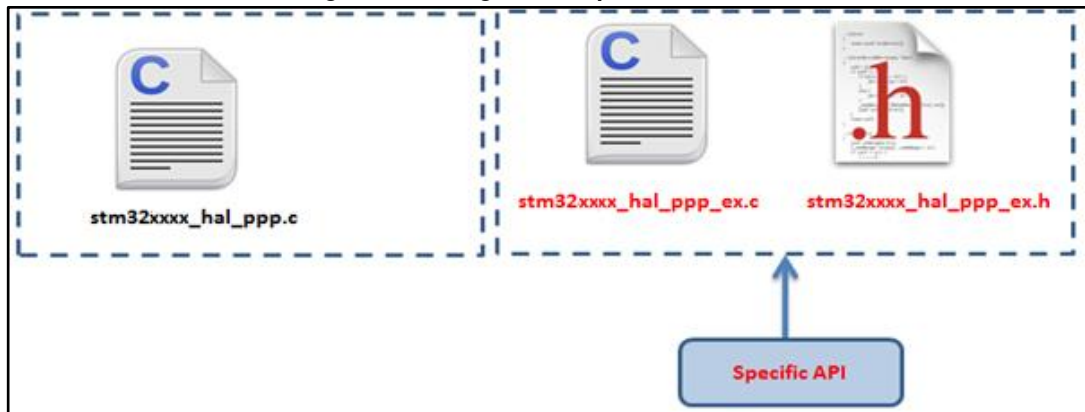
2.7.2 HAL extension model cases

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

Figure 2: Adding device-specific functions



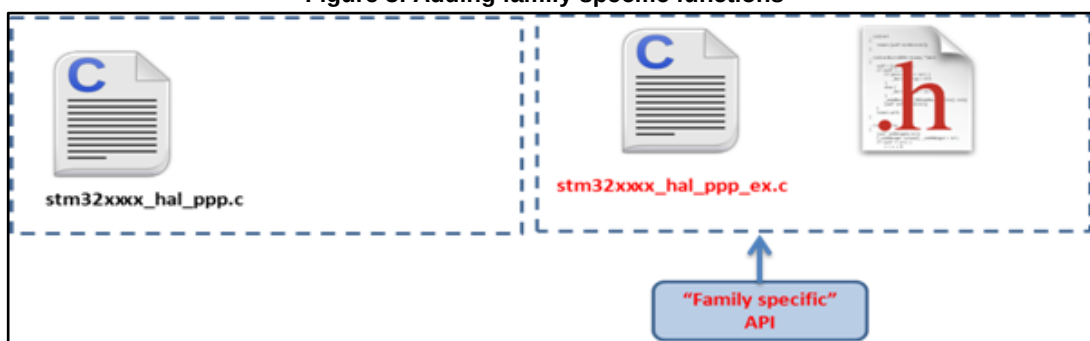
Example: stm32l0xx_hal_rcc_ex.c/h

```
#if !defined(STM32L051xx) && !defined(STM32L061xx)
void HAL_RCCEX_CRSSoftwareSynchronizationGenerate(void);
void HAL RCCEX CRSGetSynchronizationInfo(RCC CRSSynchroInfoTypeDef *pSynchroInfo);
RCC_CRSSStatusTypeDef HAL_RCCEX_CRSSWaitSynchronization(uint32_t Timeout);
#endif /* !(STM32L051xx) && !(STM32L061xx) */
```

Case2: Adding a family-specific function

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

Figure 3: Adding family-specific functions

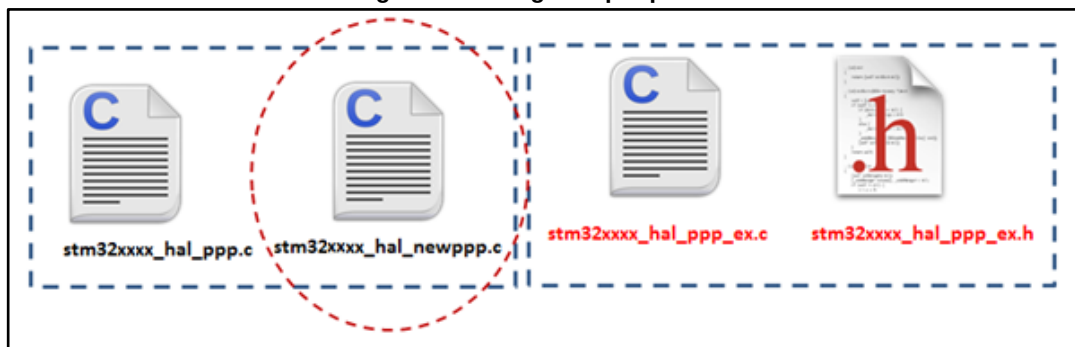


Case3: Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in `stm32l0xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32lxx_hal_conf.h` using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

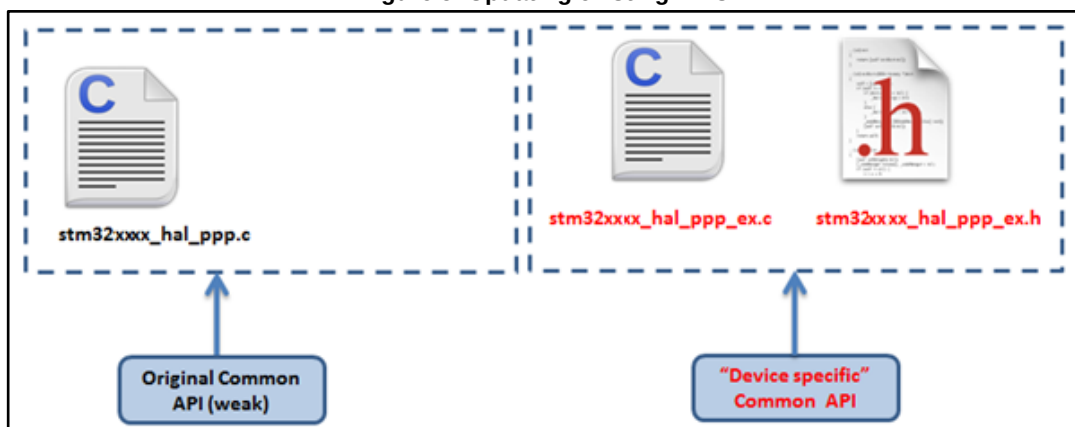


Example: `stm32l0xx_hal_lcd.c/h`

Case4: Updating existing common APIs

In this case, the routines are defined with the same names in the `stm32l0xx_hal_ppp_ex.c` extension file, while the generic API is defined as *weak*, so that the compiler will overwrite the original routine by the new defined function.

Figure 5: Updating existing APIs



Case5 : Updating existing data structures

The data structure for a specific device part number (e.g. `PPP_InitTypeDef`) can have different fields. In this case, the data structure is defined in the extension header file and delimited by the specific part number define statement.

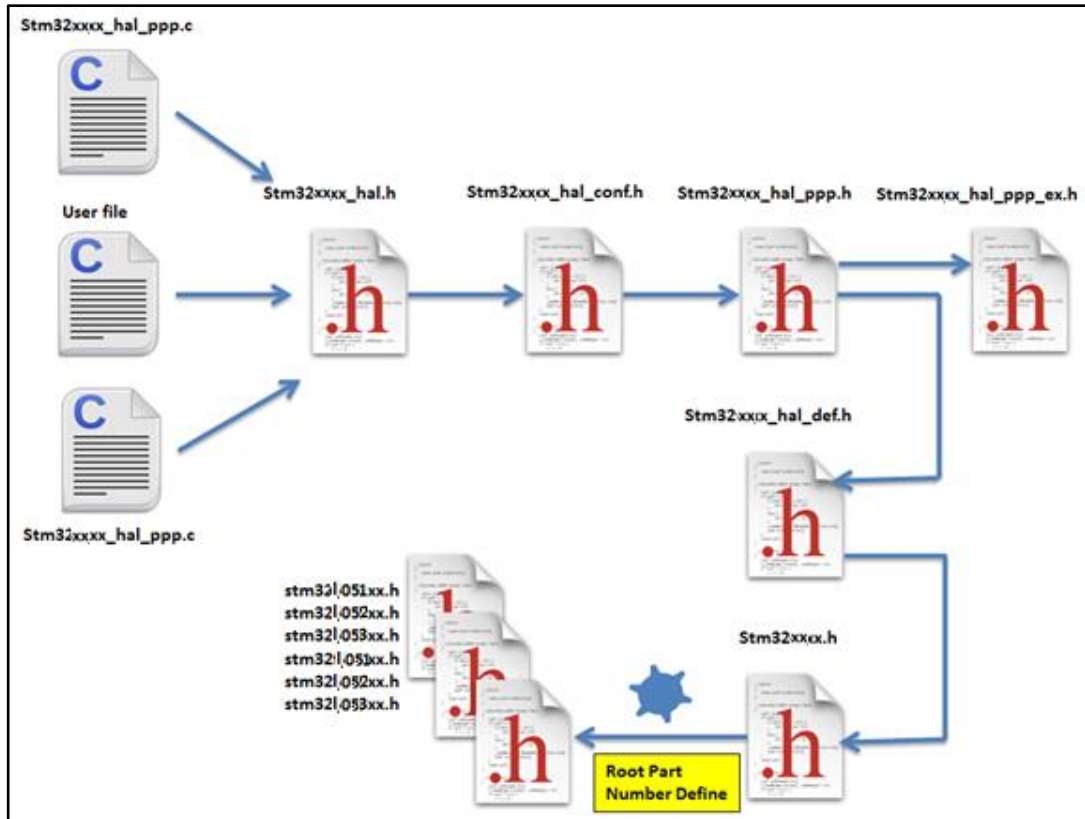
Example:

```
#if defined (STM32L051xx)
typedef struct
{
    (...)
} PPP_InitTypeDef;
#endif /* STM32L051xx */
```

2.8 File inclusion model

The header of the common HAL driver file (stm32l0xx_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 6: File inclusion model



A PPP driver is a standalone module which is used in a project. The user must enable the corresponding USE_HAL_PPP_MODULE define statement in the configuration file.

```

/*****
 * @file stm32l0xx_hal_conf.h
 * @author MCD Application Team
 * @version VX.Y.Z * @date dd-mm-yyyy
 * @brief This file contains the modules to be used
 *****/
(...)
#define USE_HAL_USART_MODULE
#define USE_HAL_IRDA_MODULE
#define USE_HAL_DMA_MODULE
#define USE_HAL_RCC_MODULE
(...)

```

2.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm32l0xx_hal_def.h*. The main common define enumeration is *HAL_StatusTypeDef*.

- **HAL Status** The HAL status is used by almost all HAL APIs, except for boolean functions and IRQ handler. It returns the status of the current API operations. It has four possible values as described below:

```
typedef enum
{
  HAL_OK = 0x00,
  HAL_ERROR = 0x01,
  HAL_BUSY = 0x02,
  HAL_TIMEOUT = 0x03
} HAL_StatusTypeDef;
```

- **HAL Locked** The HAL lock is used by all HAL APIs to prevent accessing by accident shared resources.

```
typedef enum
{
  HAL_UNLOCKED = 0x00, /*!<Resources unlocked */
  HAL_LOCKED = 0x01 /*!< Resources locked */
} HAL_LockTypeDef;
```

In addition to common resources, the *stm32l0xx_hal_def.h* file calls the *stm32l0xx.h* file in CMSIS library to get the data structures and the address mapping for all peripherals:

- Declarations of peripheral registers and bits definition.
- Macros to access peripheral registers hardware (Write register, Read register...etc.).
- **Common macros**
 - Macro defining *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, *stm32l0xx_hal_conf.h*, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

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

Table 11: Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	8 000 000 (Hz)

Configuration item	Description	Default Value
HSE_STARTUP_TIMEOUT	Timeout for HSE start up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)
MSI_VALUE	Defines the Internal Multiple Speed oscillator (MSI) value expressed in Hz.	2 000 000 (Hz)
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	TRUE
BUFFER_CACHE_ENABLE	Enables buffer cache	FALSE



The `stm32l0xx_hal_conf_template.h` file is located in the HAL drivers *Inc* folder. It should be copied to the user folder, renamed and modified as described above.



By default, the values defined in the `stm32l0xx_hal_conf_template.h` file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

2.11 HAL system peripheral handling

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

2.11.1 Clock

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

- `HAL_RCC_OscConfig` (`RCC_OscInitTypeDef *RCC_OscInitStruct`). This function configures/enables multiple clock sources (HSE, HSI, LSE, LSI, PLL).
- `HAL_RCC_ClockConfig` (`RCC_ClkInitTypeDef *RCC_ClkInitStruct`, `uint32_t FLatency`). This function
 - Selects the system clock source
 - Configures AHB, APB1 and APB2 clock dividers
 - Configures the number of Flash memory wait states
 - Updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (RTC, USB...). In this case, the clock configuration is performed by an extended API defined in `stm32l0xx_hal_ppp_ex.c`: `HAL_RCCEx_PeriphCLKConfig` (`RCC_PeriphCLKInitTypeDef *PeriphClkInit`).

Additional RCC HAL driver functions are available:

- `HAL_RCC_DeInit()` Clock de-initialization function that returns clock configuration to reset state

- Get clock functions that allow retrieving various clock configurations (system clock, HCLK, PCLK1, PCLK2, ...)
- MCO and CSS configuration functions

A set of macros are defined in stm32l0xx_hal_rcc.h. They 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()`.

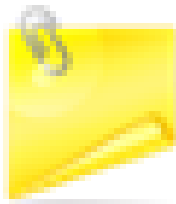
In addition to standard GPIO modes (input, output, analog), pin mode can be configured as EXTI with interrupt or event generation.

When selecting EXTI mode with interrupt generation, the user must call `HAL_GPIO_EXTI_IRQHandler()` from `stm32l0xx_it.c` and implement `HAL_GPIO_EXTI_Callback()`

The table below describes the `GPIO_InitTypeDef` structure field.

Table 12: Description of `GPIO_InitTypeDef` structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: <code>GPIO_PIN_x</code> or <code>GPIO_PIN_All</code> , where <code>x[0..15]</code>
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: • <u>GPIO mode</u> – <code>GPIO_MODE_INPUT</code> : Input Floating – <code>GPIO_MODE_OUTPUT_PP</code> : Output Push Pull – <code>GPIO_MODE_OUTPUT_OD</code> : Output Open Drain – <code>GPIO_MODE_AF_PP</code> : Alternate Function Push Pull – <code>GPIO_MODE_AF_OD</code> : Alternate Function Open Drain – <code>GPIO_MODE_ANALOG</code> : Analog mode • <u>External Interrupt Mode</u> – <code>GPIO_MODE_IT_RISING</code> : Rising edge trigger detection – <code>GPIO_MODE_IT_FALLING</code> : Falling edge trigger detection – <code>GPIO_MODE_IT_RISING_FALLING</code> : Rising/Falling edge trigger detection • <u>External Event Mode</u> – <code>GPIO_MODE_EVT_RISING</code> : Rising edge trigger detection – <code>GPIO_MODE_EVT_FALLING</code> : Falling edge trigger detection – <code>GPIO_MODE_EVT_RISING_FALLING</code> : Rising/Falling edge trigger detection

Structure field	Description
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_FREQ_LOW GPIO_SPEED_FREQ_MEDIUM GPIO_SPEED_FREQ_HIGH GPIO_SPEED_FREQ_VERY_HIGH
Alternate	Peripheral to be connected to the selected pins. Possible values: GPIO_AFx_PPP, where AFx: is the alternate function index PPP: is the peripheral instance Example: use GPIO_AF1_TIM1 to connect TIM1 IOs on AF1. These values are defined in the GPIO extended driver, since the AF mapping may change between product lines.  Refer to the “Alternate function mapping” table in the datasheets for the detailed description of the system and peripheral I/O alternate functions.

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external 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_FREQ_HIGH
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_FREQ_HIGH
GPIO_InitStruct.Alternate = GPIO_AF4_USART1;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

2.11.3 Cortex NVIC and SysTick timer

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

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

2.11.4 PWR

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

- PVD configuration, enabling/disabling and interrupt handling
 - `HAL_PWR_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()`

Depending on the STM32 Series, extension functions are available in `stm32l0xx_hal_pwr_ex`. Here are a few examples (the list is not exhaustive)

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

2.11.5 EXTI

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

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

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

Table 13: Description of EXTI configuration macros

Macros	Description
PPP_EXTI_LINE_FUNCTION	Defines the EXTI line connected to the internal peripheral. Example: #define PWR_EXTI_LINE_PVD ((uint32_t)0x00010000) /*!<External interrupt line 16 Connected to the PVD EXTI Line */
__HAL_PPP_EXTI_ENABLE_IT(__EXTI_LINE__)	Enables a given EXTI line Example: __HAL_PVD_EXTI_ENABLE_IT(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_DISABLE_IT(__EXTI_LINE__)	Disables a given EXTI line. Example: __HAL_PVD_EXTI_DISABLE_IT(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_GET_FLAG(__EXTI_LINE__)	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PVD_EXTI_GET_FLAG(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_CLEAR_FLAG(__EXTI_LINE__)	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PVD_EXTI_CLEAR_FLAG(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_GENERATE_SWIT (__EXTI_LINE__)	Generates a software interrupt for a given EXTI line. Example: __HAL_PVD_EXTI_GENERATE_SWIT (PWR_EXTI_LINE_PVD)

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32l0xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback()).

2.11.6 DMA

The DMA HAL driver allows enabling and configuring the peripheral to be connected to the DMA Channels (except for internal SRAM/FLASH memory which do not require any initialization). Refer to the product reference manual for details on the DMA request corresponding to each peripheral.

For a given stream, HAL_DMA_Init() API allows programming the required configuration through the following parameters:

- Transfer Direction
- Source and Destination data formats
- Circular, Normal or peripheral flow control mode
- Channels Priority level
- Source and Destination Increment mode
- FIFO mode and its Threshold (if needed)
- Burst mode for Source and/or Destination (if needed).

Two operating modes are available:

- Polling mode I/O operation
 - a. Use HAL_DMA_Start() to start DMA transfer when the source and destination addresses and the Length of data to be transferred have been configured.
 - b. Use HAL_DMA_PollForTransfer() to poll for the end of current transfer. In this case a fixed timeout can be configured depending on the user application.
- Interrupt mode I/O operation
 - a. Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
 - b. Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
 - c. Use HAL_DMA_Start_IT() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been configured. In this case the DMA interrupt is configured.
 - d. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
 - e. When data transfer is complete, HAL_DMA_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (i.e. a member of DMA handle structure).

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

- Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
- Use HAL_DMA_Abort() function to abort the current transfer

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: enable the specified DMA Channels.
- __HAL_DMA_DISABLE: disables the specified DMA Channels.
- __HAL_DMA_GET_FLAG: gets the DMA Channels pending flags.
- __HAL_DMA_CLEAR_FLAG: clears the DMA Channels pending flags.
- __HAL_DMA_ENABLE_IT: enables the specified DMA Channels interrupts.
- __HAL_DMA_DISABLE_IT: disables the specified DMA Channels interrupts.
- __HAL_DMA_GET_IT_SOURCE: checks whether the specified DMA stream interrupt has occurred or not.



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



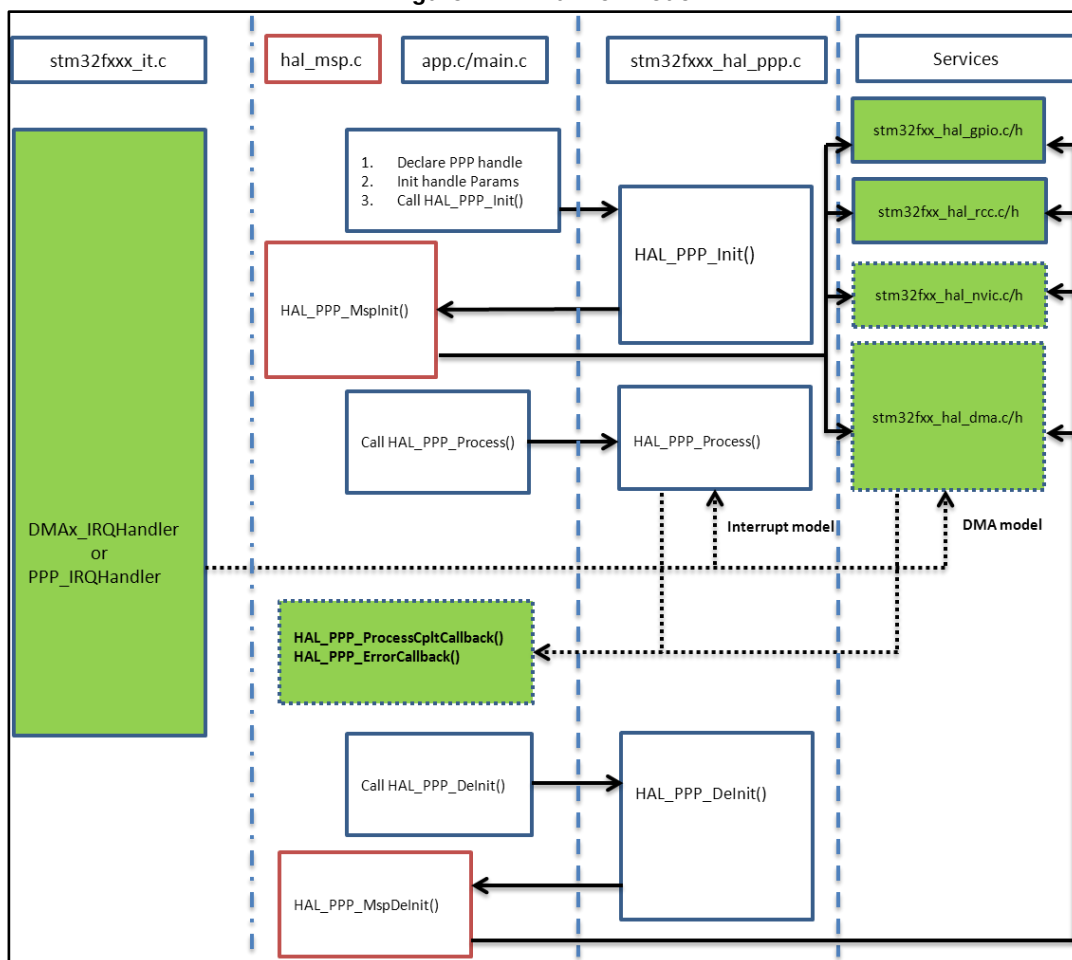
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

2.12 How to use HAL drivers

2.12.1 HAL usage models

The following figure shows the typical use of the HAL driver and the interaction between the application user, the HAL driver and the interrupts.

Figure 7: HAL driver model



Basically, the HAL driver APIs are called from user files and optionally from interrupt handlers file when the APIs based on the DMA or the PPP peripheral dedicated interrupts are used (see green blocks in the above schematics).

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

2.12.2 HAL initialization

2.12.2.1 HAL global initialization

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file `stm32l0xx_hal.c`.

- `HAL_Init()`: this function must be called at application startup to
 - Initialize data/instruction cache and pre-fetch queue
 - Set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
 - Call `HAL_MspInit()` user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). `HAL_MspInit()` is defined as “weak” empty function in the HAL drivers.
- `HAL_DeInit()`
 - Resets all peripherals
 - Calls function `HAL_MspDeInit()` which is a user callback function to do system level De-Initializations.
- `HAL_GetTick()`: this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- `HAL_Delay()`. this function implements a delay (expressed in milliseconds) using the SysTick timer.
 Care must be taken when using `HAL_Delay()` since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if `HAL_Delay()` is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be blocked.

2.12.2.2 System clock initialization

The clock configuration is done at the beginning of the user code. However the user can change the configuration of the clock in the application code. Please find below the typical Clock configuration sequence:

```
void SystemClock_Config(void)
{
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};

    /* Enable MSI Oscillator */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
    RCC_OscInitStruct.MSIState = RCC_MSI_ON;
    RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_5;
    RCC_OscInitStruct.MSICalibrationValue=0x00;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        /* Initialization Error */
        while(1);
    }
}
```

```
/* Select MSI as system clock source and configure the HCLK, PCLK1 and PCLK2
   clocks dividers */

RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);

RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
{
    /* Initialization Error */
    while(1);
}
/* Enable Power Control clock */
__HAL_RCC_PWR_CLK_ENABLE();

/* The voltage scaling allows optimizing the power consumption when the device is
   clocked below the maximum system frequency, to update the voltage scaling value
   regarding system frequency refer to product datasheet.  */
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);

/* Disable Power Control clock */
__HAL_RCC_PWR_CLK_DISABLE();
}
```

2.12.2.3 HAL MSP initialization process

The peripheral initialization is done through *HAL_PPP_Init()* while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling MSP callback function *HAL_PPP_MspInit()*.

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

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

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

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

The HAL MSP callback is implemented inside the *stm32l0xx_hal_msp.c* file in the user folders. An *stm32l0xx_hal_msp.c* file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low level initialization code.

stm32l0xx_hal_msp.c file contains the following functions:

Table 14: MSP functions

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

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

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

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

2.12.3 HAL IO operation process

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

- Polling mode
- Interrupt mode
- DMA mode

2.12.3.1 Polling mode

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

The example below shows the typical polling mode processing sequence :

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

```

2.12.3.2 Interrupt mode

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

In interrupt mode, four functions are declared in the driver:

- *HAL_PPP_Process_IT()*: launch the process
- *HAL_PPP_IRQHandler()*: the global PPP peripheral interruption
- *__weak HAL_PPP_ProcessCpltCallback ()*: the callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: the callback relative to the process Error.

To use a process in interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and *HAL_PPP_IRQHandler* in *stm32l0xx_it.c*.

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

An example of use is illustrated below:

main.c file:

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

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
}

void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
}
```

stm32l0xx_it.c file:

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

2.12.3.3 DMA mode

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

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

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *stm32l0xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
    PPP_TypeDef *Instance; /* Register base address */
    PPP_InitTypeDef Init; /* PPP communication parameters */
    HAL_StateTypeDef State; /* PPP communication state */
    (...)
    DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

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

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

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

An example of use is illustrated below:

main.c file:

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

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
}

void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
}
```

stm32l0xx_it.c file:

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

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

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

2.12.4 Timeout and error management

2.12.4.1 Timeout management

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

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

The timeout possible 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 `stm32l0xx_hal_def.h` as `0xFFFFFFFF`

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

Example of fixed timeout:

```
#define LOCAL_PROCESS_TIMEOUT 100
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef)
{
  uint32_t tickstart = 0;
  (...)
  tickstart = HAL_GetTick();
  (...)
  while(ProcessOngoing)
  {
    (...)
    if((HAL_GetTick() - tickstart) > LOCAL_PROCESS_TIMEOUT)
    {
      /* Process unlocked */
      HAL_UNLOCK(hPPP);
      hPPP->State = HAL_PPP_STATE_TIMEOUT;
      return HAL_PPP_STATE_TIMEOUT;
    }
    (...)
  }
```

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

```
HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hPPP, uint32_t Timeout)
{
    uint32_t tickstart = 0;
    (...)
    tickstart = HAL_GetTick();
    (...)
    while(ProcessOngoing)
    {
        (...)
        if(Timeout != HAL_MAX_DELAY)
        {
            if((HAL_GetTick() - tickstart) > Timeout)
            {
                /* Process unlocked */
                HAL_UNLOCK(hPPP);
                hPPP->State = HAL_PPP_STATE_TIMEOUT;
                return hPPP->State;
            }
        }
        (...)
    }
}
```

2.12.4.2 Error management

The HAL drivers implement a check for the following items:

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

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

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

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

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

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

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

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

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

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

```
(...)
/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

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

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

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

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

2.12.4.3 Run-time checking

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

To enable the run-time checking, use the *assert_param* macro, and leave the define **USE_FULL_ASSERT** uncommented in *stm32l0xx_hal_conf.h* file.

```
void HAL_UART_Init(UART_HandleTypeDef *huart)
{
  (...) /* Check the parameters */
  assert_param(IS_UART_INSTANCE(huart->Instance));
  assert_param(IS_UART_BAUDRATE(huart->Init.BaudRate));
  assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
  assert_param(IS_UART_STOPBITS(huart->Init.StopBits));
  assert_param(IS_UART_PARITY(huart->Init.Parity));
  assert_param(IS_UART_MODE(huart->Init.Mode));
  assert_param(IS_UART_HARDWARE_FLOW_CONTROL(huart->Init.HwFlowCtl));
  (...)

  /** @defgroup UART_Word_Length *
  @{
  */
  #define UART_WORDLENGTH_8B ((uint32_t)0x00000000)
  #define UART_WORDLENGTH_9B ((uint32_t)USART_CR1_M)
  #define IS_UART_WORD_LENGTH(LENGTH) (((LENGTH) == UART_WORDLENGTH_8B) || \
    \ ((LENGTH) == UART_WORDLENGTH_9B))
```

If the expression passed to the *assert_param* macro is false, the *assert_failed* function is called and returns the name of the source file and the source line number of the call that failed. If the expression is true, no value is returned.

The *assert_param* macro is implemented in *stm32l0xx_hal_conf.h*:

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

```
#else
#define assert_param(expr) ((void)0)
#endif /* USE_FULL_ASSERT */
```

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

```
#ifndef 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:

- Common HAL APIs
- Services HAL APIs

3.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [*HAL_Init\(\)*](#)
- [*HAL_DeInit\(\)*](#)
- [*HAL_MspInit\(\)*](#)
- [*HAL_MspDeInit\(\)*](#)
- [*HAL_InitTick\(\)*](#)

3.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier

- Get the device revision identifier
- Configure low power mode behavior when the MCU is in Debug mode
- Manage the VREFINT feature (activation, lock, output selection)

This section contains the following APIs:

- [*HAL_IncTick\(\)*](#)
- [*HAL_GetTick\(\)*](#)
- [*HAL_Delay\(\)*](#)
- [*HAL_SuspendTick\(\)*](#)
- [*HAL_ResumeTick\(\)*](#)
- [*HAL_GetHalVersion\(\)*](#)
- [*HAL_GetREVID\(\)*](#)
- [*HAL_GetDEVID\(\)*](#)
- [*HAL_DBGMCU_EnableDBGSleepMode\(\)*](#)
- [*HAL_DBGMCU_DisableDBGSleepMode\(\)*](#)
- [*HAL_DBGMCU_EnableDBGStopMode\(\)*](#)
- [*HAL_DBGMCU_DisableDBGStopMode\(\)*](#)
- [*HAL_DBGMCU_EnableDBGStandbyMode\(\)*](#)
- [*HAL_DBGMCU_DisableDBGStandbyMode\(\)*](#)
- [*HAL_DBGMCU_DBG_EnableLowPowerConfig\(\)*](#)
- [*HAL_DBGMCU_DBG_DisableLowPowerConfig\(\)*](#)
- [*HAL_SYSCFG_GetBootMode\(\)*](#)
- [*HAL_SYSCFG_EnableVREFINT\(\)*](#)
- [*HAL_SYSCFG_DisableVREFINT\(\)*](#)
- [*HAL_SYSCFG_VREFINT_OutputSelect\(\)*](#)
- [*HAL_SYSCFG_Enable_Lock_VREFINT\(\)*](#)
- [*HAL_SYSCFG_Disable_Lock_VREFINT\(\)*](#)

3.1.4 HAL_Init

Function Name	HAL_StatusTypeDef HAL_Init (void)
Function Description	This function configures the Flash prefetch, Flash preread and Buffer cache, Configures time base source, NVIC and Low level hardware.
Return values	• HAL: status
Notes	• This function is called at the beginning of program after reset and before the clock configuration • The time base configuration is based on MSI clock when exiting from Reset. Once done, time base tick start incrementing. In the default implementation, SysTick is used as source of time base. the tick variable is incremented each 1ms in its ISR.

3.1.5 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	• HAL: status
Notes	• This function is optional.

3.1.6 HAL_MspInit

Function Name **void HAL_MspInit (void)**

Function Description Initializes the MSP.

Return values

- **None:**

3.1.7 HAL_MspDeInit

Function Name **void HAL_MspDeInit (void)**

Function Description DeInitializes the MSP.

Return values

- **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

- **TickPriority:** Tick interrupt priority.

Return values

- **HAL:** status

Notes

- 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

- **None:**

Notes

- 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

- **tick:** value

Notes

- 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 ms) based on a variable incremented.
Parameters	Delay: specifies the delay time length, in milliseconds.
Return values	None:
Notes	- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented. - This function is declared as <code>__weak</code> to be overwritten in case of other implementations in user file.

3.1.12 HAL_SuspendTick

Function Name	void HAL_SuspendTick (void)
Function Description	Suspends the Tick increment.
Return values	None:
Notes	- 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 <code>__weak</code> to be overwritten in case of other implementations in user file.

3.1.13 HAL_ResumeTick

Function Name	void HAL_ResumeTick (void)
Function Description	Resumes the Tick increment.
Return values	None:
Notes	- 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 <code>__weak</code> 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	Returns the HAL revision.
Return values	version: 0xXYZR (8bits for each decimal, R for RC)

3.1.15 HAL_GetREVID

Function Name	uint32_t HAL_GetREVID (void)

	Function Description	Returns the device revision identifier.
	Return values	• Device: revision identifier
3.1.16	HAL_GetDEVID	
	Function Name	uint32_t HAL_GetDEVID (void)
	Function Description	Returns the device identifier.
	Return values	• Device: identifier
3.1.17	HAL_DBGMCU_EnableDBGSleepMode	
	Function Name	void HAL_DBGMCU_EnableDBGSleepMode (void)
	Function Description	Enables the Debug Module during SLEEP mode.
	Return values	• None:
3.1.18	HAL_DBGMCU_DisableDBGSleepMode	
	Function Name	void HAL_DBGMCU_DisableDBGSleepMode (void)
	Function Description	Disables the Debug Module during SLEEP mode.
	Return values	• None:
3.1.19	HAL_DBGMCU_EnableDBGStopMode	
	Function Name	void HAL_DBGMCU_EnableDBGStopMode (void)
	Function Description	Enables the Debug Module during STOP mode.
	Return values	• None:
3.1.20	HAL_DBGMCU_DisableDBGStopMode	
	Function Name	void HAL_DBGMCU_DisableDBGStopMode (void)
	Function Description	Disables the Debug Module during STOP mode.
	Return values	• None:
3.1.21	HAL_DBGMCU_EnableDBGStandbyMode	
	Function Name	void HAL_DBGMCU_EnableDBGStandbyMode (void)
	Function Description	Enables the Debug Module during STANDBY mode.
	Return values	• None:
3.1.22	HAL_DBGMCU_DisableDBGStandbyMode	
	Function Name	void HAL_DBGMCU_DisableDBGStandbyMode (void)
	Function Description	Disables the Debug Module during STANDBY mode.
	Return values	• None:
3.1.23	HAL_DBGMCU_DBG_EnableLowPowerConfig	

Function Name	void HAL_DBGMCU_DBG_EnableLowPowerConfig (uint32_t Periph)
Function Description	Enable low power mode behavior when the MCU is in Debug mode.
Parameters	• Periph: specifies the low power mode. This parameter can be any combination of the following values: DBGMCU_SLEEP: Keep debugger connection during SLEEP mode DBGMCU_STOP: Keep debugger connection during STOP mode DBGMCU_STANDBY: Keep debugger connection during STANDBY mode
Return values	• None:

3.1.24 HAL_DBGMCU_DBG_DisableLowPowerConfig

Function Name	void HAL_DBGMCU_DBG_DisableLowPowerConfig (uint32_t Periph)
Function Description	Disable low power mode behavior when the MCU is in Debug mode.
Parameters	• Periph: specifies the low power mode. This parameter can be any combination of the following values: DBGMCU_SLEEP: Keep debugger connection during SLEEP mode DBGMCU_STOP: Keep debugger connection during STOP mode DBGMCU_STANDBY: Keep debugger connection during STANDBY mode
Return values	• None:

3.1.25 HAL_SYSCFG_GetBootMode

Function Name	uint32_t HAL_SYSCFG_GetBootMode (void)
Function Description	Returns the boot mode as configured by user.
Return values	• The: boot mode as configured by user. The returned value can be one of the following values: 0x00000000 : Boot is configured in Main Flash memory 0x00000100 : Boot is configured in System Flash memory 0x00000300 : Boot is configured in Embedded SRAM memory

3.1.26 HAL_SYSCFG_EnableVREFINT

Function Name	void HAL_SYSCFG_EnableVREFINT (void)
Function Description	Enables the VREFINT.
Return values	• None:

3.1.27 HAL_SYSCFG_DisableVREFINT

Function Name	void HAL_SYSCFG_DisableVREFINT (void)
Function Description	Disables the VREFINT.
Return values	• None:

3.1.28 HAL_SYSCFG_VREFINT_OutputSelect

Function Name **void HAL_SYSCFG_VREFINT_OutputSelect (uint32_t SYSCFG_Vrefint_OUTPUT)**

Function Description Selects the output of internal reference voltage (VREFINT).

Parameters

- **SYSCFG_Vrefint_OUTPUT:** new state of the Vrefint output. This parameter can be one of the following values:
SYSCFG_VREFINT_OUT_NONESYSCFG_VREFINT_OUT_PB0SYSCFG_VREFINT_OUT_PB1SYSCFG_VREFINT_OUT_PB0_PB1

Return values

- **None:**

3.1.29 HAL_SYSCFG_Enable_Lock_VREFINT

Function Name **void HAL_SYSCFG_Enable_Lock_VREFINT (void)**

Function Description Lock the SYSCFG VREF register values.

Return values

- **None:**

3.1.30 HAL_SYSCFG_Disable_Lock_VREFINT

Function Name **void HAL_SYSCFG_Disable_Lock_VREFINT (void)**

Function Description Unlock the overall SYSCFG VREF register values.

Return values

- **None:**

3.2 HAL Firmware driver defines**3.2.1 HAL*****DBGMCU Low Power Configuration***

DBGMCU_SLEEP

DBGMCU_STOP

DBGMCU_STANDBY

IS_DBGMCU_PERIPH

HAL Exported Macros

__HAL_DBGMCU_FREEZE_TIM2

TIM2 Peripherals Debug mode

__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_RTC
__HAL_DBGMCU_UNFREEZE_RTC
__HAL_DBGMCU_FREEZE_WWDG
__HAL_DBGMCU_UNFREEZE_WWDG
__HAL_DBGMCU_FREEZE_IWDG
__HAL_DBGMCU_UNFREEZE_IWDG
__HAL_DBGMCU_FREEZE_I2C1_TIMEOUT
__HAL_DBGMCU_UNFREEZE_I2C1_TIMEOUT
__HAL_DBGMCU_FREEZE_I2C2_TIMEOUT
__HAL_DBGMCU_UNFREEZE_I2C2_TIMEOUT
__HAL_DBGMCU_FREEZE_I2C3_TIMEOUT
__HAL_DBGMCU_UNFREEZE_I2C3_TIMEOUT
__HAL_DBGMCU_FREEZE_LPTIMER
__HAL_DBGMCU_UNFREEZE_LPTIMER
__HAL_DBGMCU_FREEZE_TIM22
__HAL_DBGMCU_UNFREEZE_TIM22
__HAL_DBGMCU_FREEZE_TIM21
__HAL_DBGMCU_UNFREEZE_TIM21
__HAL_SYSCFG_REMAPMEMORY_FLASH
__HAL_SYSCFG_REMAPMEMORY_SYSTEMFLASH
__HAL_SYSCFG_REMAPMEMORY_SRAM
__HAL_SYSCFG_DBG_LP_CONFIG

```

Description:

- Configuration of the DBG Low Power mode.

Parameters:

- `__DBG_LPMODE__`: bit field to indicate in which Low Power mode DBG is still active. This parameter can be a value of
 - `DBGMCU_SLEEP`
 - `DBGMCU_STOP`
 - `DBGMCU_STANDBY`

```
__HAL_SYSCFG_GET_BOOT_MODE
```

Description:

- Returns the boot mode as

configured by user.

Return value:

- The: boot mode as configured by user. The returned can be a value of :
 - SYSCFG_BOOT_MAINFLASH
 - SYSCFG_BOOT_SYSTEMFLASH
 - SYSCFG_BOOT_SRAM

Description:

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

Parameters:

- `__FLAG__` : specifies the flag to check. The only parameter supported is SYSCFG_FLAG_VREFINT_READY

Return value:

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

`__HAL_SYSCFG_GET_FLAG`

`__HAL_SYSCFG_FASTMODEPLUS_ENABLE`

Description:

- Fast mode Plus driving capability enable macro.

Parameters:

- `__FASTMODEPLUS__`: This parameter can be a value of :
 - SYSCFG_FASTMODEPLUS_PB6
 - SYSCFG_FASTMODEPLUS_PB7
 - SYSCFG_FASTMODEPLUS_PB8
 - SYSCFG_FASTMODEPLUS_PB9

`__HAL_SYSCFG_FASTMODEPLUS_DISABLE`

Description:

- Fast mode Plus driving capability disable macro.

Parameters:

- `__FASTMODEPLUS__`: This parameter can be a value of :
 - SYSCFG_FASTMODEPLUS_PB6
 - SYSCFG_FASTMODEPLUS_PB7

- SYSCFG_FASTMODEPLUS_PB8
- SYSCFG_FASTMODEPLUS_PB9

HAL Version

__STM32L0xx_HAL_VERSION_MAIN [31:24] main version
 __STM32L0xx_HAL_VERSION_SUB1 [23:16] sub1 version
 __STM32L0xx_HAL_VERSION_SUB2 [15:8] sub2 version
 __STM32L0xx_HAL_VERSION_RC [7:0] release candidate
 __STM32L0xx_HAL_VERSION
 IDCODE_DEVID_MASK

Boot Mode

SYSCFG_BOOT_MAINFLASH
 SYSCFG_BOOT_SYSTEMFLASH
 SYSCFG_BOOT_SRAM

Fast Mode Plus on GPIO

SYSCFG_FASTMODEPLUS_PB6
 SYSCFG_FASTMODEPLUS_PB7
 SYSCFG_FASTMODEPLUS_PB8
 SYSCFG_FASTMODEPLUS_PB9
 IS_SYSCFG_FASTMODEPLUS

SYSCFG Flags Definition

SYSCFG_FLAG_VREFINT_READY
 IS_SYSCFG_FLAG

SYSCFG LCD External Capacitors

SYSCFG_LCD_EXT_CAPA	Connection of internal Vlcd rail to external capacitors
SYSCFG_VLCD_PB2_EXT_CAPA_ON	Connection on PB2
SYSCFG_VLCD_PB12_EXT_CAPA_ON	Connection on PB12
SYSCFG_VLCD_PE11_EXT_CAPA_ON	Connection on PB0
SYSCFG_VLCD_PB0_EXT_CAPA_ON	Connection on PE11
SYSCFG_VLCD_PE12_EXT_CAPA_ON	Connection on PE12

SYSCFG VREFINT Out Selection

SYSCFG_VREFINT_OUT_NONE
 SYSCFG_VREFINT_OUT_PB0
 SYSCFG_VREFINT_OUT_PB1
 SYSCFG_VREFINT_OUT_PB0_PB1
 IS_SYSCFG_VREFINT_OUT_SELECT

4 HAL ADC Generic Driver

4.1 ADC Firmware driver registers structures

4.1.1 ADC_OversamplingTypeDef

Data Fields

- *uint32_t Ratio*
- *uint32_t RightBitShift*
- *uint32_t TriggeredMode*

Field Documentation

- *uint32_t ADC_OversamplingTypeDef::Ratio*
Configures the oversampling ratio. This parameter can be a value of [ADC_Oversampling_Ratio](#)
- *uint32_t ADC_OversamplingTypeDef::RightBitShift*
Configures the division coefficient for the Oversampler. This parameter can be a value of [ADC_Right_Bit_Shift](#)
- *uint32_t ADC_OversamplingTypeDef::TriggeredMode*
Selects the regular triggered oversampling mode This parameter can be a value of [ADC_Triggered_Oversampling_Mode](#)

4.1.2 ADC_InitTypeDef

Data Fields

- *uint32_t OversamplingMode*
- *ADC_OversamplingTypeDef Oversample*
- *uint32_t ClockPrescaler*
- *uint32_t Resolution*
- *uint32_t SamplingTime*
- *uint32_t ScanConvMode*
- *uint32_t DataAlign*
- *uint32_t ContinuousConvMode*
- *uint32_t DiscontinuousConvMode*
- *uint32_t ExternalTrigConv*
- *uint32_t ExternalTrigConvEdge*
- *uint32_t DMAContinuousRequests*
- *uint32_t EOCSelection*
- *uint32_t Overrun*
- *uint32_t LowPowerAutoWait*
- *uint32_t LowPowerFrequencyMode*
- *uint32_t LowPowerAutoPowerOff*

Field Documentation

- **`uint32_t ADC_InitTypeDef::OversamplingMode`**
Specifies whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- **`ADC_OversamplingTypeDef ADC_InitTypeDef::Oversample`**
Specifies the Oversampling parameters. Note: This parameter can be modified only if there is no conversion is ongoing.
- **`uint32_t ADC_InitTypeDef::ClockPrescaler`**
Selects the ADC clock frequency. This parameter can be a value of [ADC_ClockPrescaler](#). Note: This parameter can be modified only if ADC is disabled. Note: In case of Synchronous clock mode divided by 1, this configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle). Refer to reference manual for details.
- **`uint32_t ADC_InitTypeDef::Resolution`**
Configures the ADC resolution mode. This parameter can be a value of [ADC_Resolution](#). Note: This parameter can be modified only if ADC is disabled.
- **`uint32_t ADC_InitTypeDef::SamplingTime`**
The sample time value to be set for all channels. This parameter can be a value of [ADC_sampling_times](#). Note: This parameter can be modified only if there is no conversion ongoing.
- **`uint32_t ADC_InitTypeDef::ScanConvMode`**
The scan sequence direction. 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, ...). This parameter can be a value of [ADC_Scan_mode](#). Note: This parameter can be modified only if there is no conversion is ongoing.
- **`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](#). Note: This parameter can be modified only if there is no conversion is ongoing.
- **`uint32_t ADC_InitTypeDef::ContinuousConvMode`**
Specifies whether the conversion is performed in Continuous or Single mode. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- **`uint32_t ADC_InitTypeDef::DiscontinuousConvMode`**
Specifies whether the conversion is performed in Complete-sequence/Discontinuous-sequence. Discontinuous mode can be enabled only if continuous mode is disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.
- **`uint32_t ADC_InitTypeDef::ExternalTrigConv`**
Select the external event used to trigger the start of conversion. If set to ADC_SOFTWARE_START, external triggers are disabled. This parameter can be a value of [ADC_External_trigger_Source](#). Note: This parameter can be modified only if there is no conversion is ongoing.
- **`uint32_t ADC_InitTypeDef::ExternalTrigConvEdge`**
Select the external trigger edge and enable the trigger. If trigger is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of [ADC_Regular_External_Trigger_Source_Edge](#). Note: This parameter can be modified only if there is no conversion is ongoing.
- **`uint32_t ADC_InitTypeDef::DMAContinuousRequests`**
Specifies whether the DMA requests are performed in one shot mode (DMA transfer stop when number of conversions is reached) or in Continuous mode (DMA transfer unlimited, whatever number of conversions). Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer

max pointer is reached. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing.

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

4.1.3 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

4.1.4 ADC_ChannelConfTypeDef

Data Fields

- `uint32_t Channel`
- `uint32_t Rank`

Field Documentation

- `uint32_t ADC_ChannelConfTypeDef::Channel`
the ADC channel to configure This parameter can be a value of [ADC_channels](#)
- `uint32_t ADC_ChannelConfTypeDef::Rank`
Add or remove the channel from ADC regular group sequencer. On STM32L0 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](#)

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

- **`uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold`**
Configures the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between `Min_Data = 0x000` and `Max_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F` respectively.

4.2 ADC Firmware driver API description

4.2.1 ADC peripheral features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
- A built-in hardware oversampler can handle multiple conversions and average them into a single data with increased data width, up to 16-bit.
- 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 STM32L0, ADC clock frequency max is 16MHz (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 16MHz. If asynchronous clock is selected, parameter "HSIState" must be set either: - to "...HSIState = RCC_HSI_ON" to maintain the HSI16 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) HSI16 enable : (optional: if asynchronous clock selected)

- `RCC_OscInitTypeDef` `RCC_OscInitStructure`;
- `RCC_OscInitStructure.OscillatorType` = `RCC_OSCILLATORTYPE_HSI`;
- `RCC_OscInitStructure.HSI16CalibrationValue` = `RCC_HSICALIBRATION_DEFAULT`;
- `RCC_OscInitStructure.HSIState` = `RCC_HSI_ON`;
- `RCC_OscInitStructure.PLL...` (optional if used for system clock)
- `HAL_RCC_OscConfig(&RCC_OscInitStructure)`;
- 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, oversampler, continuous mode, ...) 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()`.
4. When device is in mode low-power (low-power run, low-power sleep or stop mode), function "`HAL_ADCEx_EnableVREFINT()`" must be called before function `HAL_ADC_Init()`. In case of internal temperature sensor to be measured: function "`HAL_ADCEx_EnableVREFINTTempSensor()`" must be called similarly

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_MspDeInit()` (recommended code location) or with other device clock parameters configuration:
 - `RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI;`
 - `RCC_OscInitStructure.HSIState = RCC_HSI_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_DeInit()`.
 - Disable the NVIC for DMA using function `HAL_NVIC_DisableIRQ(DMAx_Channelx_IRQn)`

4.2.3 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\(\)*](#)

4.2.4 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\(\)*](#)

4.2.5 ADC Peripheral State functions

This subsection provides functions allowing to

- Check the ADC state.
- handle ADC interrupt request.

This section contains the following APIs:

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

4.2.6 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	• hadc: ADC handle
Return values	• HAL: status
Notes	• As prerequisite, ADC clock must be configured at RCC top level depending on both possible clock sources: APB clock of 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". • When device is in mode low-power (low-power run, low-power sleep or stop mode), function "HAL_ADCEx_EnableVREFINT()" must be called before function HAL_ADC_Init() (in case of previous ADC operations: function HAL_ADC_DeInit() must be called first). In case of internal temperature sensor to be measured: function "HAL_ADCEx_EnableVREFINTTempSensor()" must be called similarly.

4.2.7 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	• hadc: ADC handle
Return values	• 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.

4.2.8 HAL_ADC_MspInit

	Function Name	void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)
	Function Description	Initializes the ADC MSP.
	Parameters	• hadc: ADC handle
	Return values	• None:
4.2.9	HAL_ADC_MspDeInit	
	Function Name	void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)
	Function Description	DeInitializes the ADC MSP.
	Parameters	• hadc: ADC handle
	Return values	• None:
4.2.10	HAL_ADC_Start	
	Function Name	HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)
	Function Description	Enables ADC, starts conversion of regular group.
	Parameters	• hadc: ADC handle
	Return values	• HAL: status
4.2.11	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	• hadc: ADC handle
	Return values	• HAL: status.
4.2.12	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	• hadc: ADC handle • Timeout: Timeout value in millisecond.
	Return values	• HAL: status
	Notes	• 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).

4.2.13 HAL_ADC_PollForEvent

Function Name	HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)
Function Description	Poll for conversion event.
Parameters	• hadc: ADC handle • EventType: the ADC event type. This parameter can be one of the following values: ADC_AWD_EVENT: ADC Analog watchdog event ADC_OVR_EVENT: ADC Overrun event • Timeout: Timeout value in millisecond.
Return values	• HAL: status

4.2.14 HAL_ADC_Start_IT

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

4.2.15 HAL_ADC_Stop_IT

Function Name	HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)
Function Description	Stop ADC conversion of regular group, disable interruption of end-of-conversion, disable ADC peripheral.
Parameters	• hadc: ADC handle
Return values	• HAL: status.

4.2.16 HAL_ADC_Start_DMA

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

4.2.17 HAL_ADC_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)
Function Description	Stop ADC conversion of regular group, disable ADC DMA transfer, disable ADC peripheral.
Parameters	• hadc: ADC handle
Return values	• HAL: status.

4.2.18 HAL_ADC_GetValue

Function Name	uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)
Function Description	Get ADC regular group conversion result.
Parameters	hadc: ADC handle
Return values	Converted: value
Notes	- Reading DR register automatically clears EOC (end of conversion of regular group) flag. - This function does not clear ADC flag EOS (ADC group regular end of sequence conversion). Occurrence of flag EOS rising: If sequencer is composed of 1 rank, flag EOS is equivalent to flag EOC. If sequencer is composed of several ranks, during the scan sequence flag EOC only is raised, at the end of the scan sequence both flags EOC and EOS are raised. To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADC_PollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS).

4.2.19 HAL_ADC_IRQHandler

Function Name	void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)
Function Description	Handles ADC interrupt request.
Parameters	hadc: ADC handle
Return values	None:

4.2.20 HAL_ADC_ConvCpltCallback

Function Name	void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)
Function Description	Conversion complete callback in non blocking mode.
Parameters	hadc: ADC handle
Return values	None:

4.2.21 HAL_ADC_ConvHalfCpltCallback

Function Name	void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)
Function Description	Conversion DMA half-transfer callback in non blocking mode.
Parameters	hadc: ADC handle
Return values	None:

4.2.22 HAL_ADC_LevelOutOfWindowCallback

Function Name	void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)
Function Description	Analog watchdog callback in non blocking mode.

Parameters	• hadc: ADC handle
Return values	• None:

4.2.23 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	• hadc: ADC handle
Return values	• None:

4.2.24 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	• hadc: ADC handle • sConfig: Structure of ADC channel for regular group.
Return values	• HAL: status
Notes	• In case of usage of internal measurement channels: VrefInt/Vlcd (STM32L0x3xx only)/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_vlcd (STM32L0x3xx only), TS_temp (values rough order: 5us to 17us). These internal paths can be be disabled using function HAL_ADC_DeInit(). • Possibility to update parameters on the fly: This function initializes channel into regular group, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_ChannelConfTypeDef".

4.2.25 HAL_ADC_AnalogWDGConfig

Function Name	HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)
Function Description	Configures the analog watchdog.
Parameters	• hadc: ADC handle • AnalogWDGConfig: Structure of ADC analog watchdog configuration
Return values	• HAL: status
Notes	• 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".

4.2.26 HAL_ADC_GetState

Function Name	uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)
Function Description	return the ADC state
Parameters	• hadc: ADC handle
Return values	• HAL: state

4.2.27 HAL_ADC_GetError

Function Name	uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)
Function Description	Return the ADC error code.
Parameters	• hadc: ADC handle
Return values	• 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_18
 ADC_CHANNEL_VLCD
 ADC_CHANNEL_VREFINT
 ADC_CHANNEL_TEMPSENSOR

ADC Channel Masks

ADC_CHANNEL_MASK
 ADC_CHANNEL_AWD_MASK

ADC Clock Prescaler

ADC_CLOCK_ASYNC_DIV1	ADC Asynchronous clock mode divided by 1
ADC_CLOCK_ASYNC_DIV2	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV4	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV6	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV8	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV10	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV12	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV16	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV32	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV64	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV128	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_ASYNC_DIV256	ADC Asynchronous clock mode divided by 2
ADC_CLOCK_SYNC_PCLK_DIV1	Synchronous clock mode divided by 1 This configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle)
ADC_CLOCK_SYNC_PCLK_DIV2	Synchronous clock mode divided by 2
ADC_CLOCK_SYNC_PCLK_DIV4	Synchronous clock mode divided by 4

ADC Conversion Group

ADC_REGULAR_GROUP

ADC Data Align

ADC_DATAALIGN_RIGHT
 ADC_DATAALIGN_LEFT

ADC EOC Selection

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 OVR error

HAL_ADC_ERROR_DMA DMA transfer error

ADC Event

ADC_AWD_EVENT

ADC_OVR_EVENT

ADC Exported Macro

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_ENABLE

Description:

- Enable the ADC peripheral.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

ADC_ENABLING_CONDITIONS

Description:

- Verification of hardware constraints before ADC can be enabled.

Parameters:

- __HANDLE__: ADC handle

Return value:

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

__HAL_ADC_DISABLE

Description:

- Disable the ADC peripheral.

ADC_DISABLING_CONDITIONS**Parameters:**

- `__HANDLE__`: ADC handle

Return value:

- None

Description:

- Verification of hardware constraints before ADC can be disabled.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

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

Description:

- Verification of ADC state: enabled or disabled.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

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

Description:

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

Parameters:

- `__HANDLE__`: ADC handle

Return value:

- None

Description:

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

Parameters:

- `__HANDLE__`: ADC handle

Return value:

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

ADC_IS_CONVERSION_ONGOING_REGULAR**Description:**

- Check if no conversion on going

on regular group.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

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

Description:

- Enable ADC continuous conversion mode.

Parameters:

- `_CONTINUOUS_MODE_`: Continuous mode.

Return value:

- None

Description:

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

Parameters:

- `_SCAN_MODE_`: Scan conversion mode.

Return value:

- None

Description:

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

Parameters:

- `_NBR_DISCONTINUOUS_CONV_`: Number of discontinuous conversions.

Return value:

- None

Description:

- Enable the ADC DMA continuous request.

Parameters:

- `_DMAContReq_MODE_`: DMA continuous request mode.

Return value:

ADC_CONTINUOUS

ADC_SCANDIR

`__HAL_ADC_CFGR1_DISCONTINUOUS_NUM`

ADC_DMACONTREQ

__HAL_ADC_CFGR1_AutoDelay

- None

Description:

- Enable the ADC Auto Delay.

Parameters:

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

Return value:

- None

__HAL_ADC_CFGR1_AUTOFF

Description:

- Enable the ADC LowPowerAutoPowerOff.

Parameters:

- `_AUTOFF_`: AutoOff bit enable or disable.

Return value:

- None

ADC_TRX_HIGHTHRESHOLD

Description:

- Configure the analog watchdog high threshold into registers TR1, TR2 or TR3.

Parameters:

- `_Threshold_`: Threshold value

Return value:

- None

__HAL_ADC_CCR_LOWFREQUENCY

Description:

- Enable the ADC Low Frequency mode.

Parameters:

- `_LOW_FREQUENCY_MODE_`: Low Frequency mode.

Return value:

- None

ADC_OFFSET_SHIFT_RESOLUTION

Description:

- Shift the offset in function of the selected ADC resolution.

Parameters:

- `__HANDLE__`: ADC handle.
- `_Offset_`: Value to be shifted

Return value:

ADC_AWD1THRESHOLD_SHIFT_RESOLUTION

- None

Description:

- Shift the AWD1 threshold in function of the selected ADC resolution.

Parameters:

- `__HANDLE__`: ADC handle.
- `__Threshold_`: Value to be shifted

Return value:

- None

`__HAL_ADC_Value_Shift_left`

Description:

- Shift the value on the left, less significant are set to 0.

Parameters:

- `__Value_`: Value to be shifted
- `__Shift_`: Number of shift to be done

Return value:

- None

`__HAL_ADC_ENABLE_IT`

Description:

- Enable the ADC end of conversion interrupt.

Parameters:

- `__HANDLE__`: ADC handle.
- `__INTERRUPT__`: ADC Interrupt.

Return value:

- None

`__HAL_ADC_DISABLE_IT`

Description:

- Disable the ADC end of conversion interrupt.

Parameters:

- `__HANDLE__`: ADC handle.
- `__INTERRUPT__`: ADC interrupt.

Return value:

- None

`__HAL_ADC_GET_IT_SOURCE`

Description:

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

Parameters:

- `__HANDLE__`: ADC handle

`__HAL_ADC_CLEAR_FLAG`

- `__INTERERRUPT__`: ADC interrupt source to check
- ...
- ...

Return value:

- State: of interruption (TRUE or FALSE)

Description:

- Clear the ADC's pending flags.

Parameters:

- `__HANDLE__`: ADC handle.
- `__FLAG__`: ADC flag.

Return value:

- None

Description:

- Get the selected ADC's flag status.

Parameters:

- `__HANDLE__`: ADC handle.
- `__FLAG__`: ADC flag.

Return value:

- None

Description:

- Simultaneously clears and sets specific bits of the handle State.

Return value:

- None

Notes:

- `: ADC_STATE_CLR_SET()` macro is merely aliased to generic macro `MODIFY_REG()`, the first parameter is the ADC handle State, the second parameter is the bit field to clear, the third and last parameter is the bit field to set.

Description:

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

Parameters:

- `__HANDLE__`: ADC handle

Return value:

`ADC_STATE_CLR_SET`

`ADC_CLEAR_ERRORCODE`

__HAL_ADC_CLOCK_PRESCALER

- None

Description:

- Configuration of ADC clock & prescaler: clock source PCLK or Asynchronous with selectable prescaler.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

IS_ADC_CLOCKPRESCALER

IS_ADC_RESOLUTION

IS_ADC_RESOLUTION_8_6_BITS

IS_ADC_DATA_ALIGN

IS_ADC_EXTTRIG_EDGE

IS_ADC_EOC_SELECTION

IS_ADC_OVERRUN

IS_ADC_RANK

IS_ADC_CHANNEL

IS_ADC_SAMPLE_TIME

IS_ADC_SCAN_MODE

IS_ADC_OVERSAMPLING_RATIO

IS_ADC_RIGHT_BIT_SHIFT

IS_ADC_TRIGGERED_OVERSAMPLING_MODE

IS_ADC_ANALOG_WATCHDOG_MODE

IS_ADC_CONVERSION_GROUP

IS_ADC_EVENT_TYPE

ADC Exported Types

HAL_ADC_STATE_RESET

ADC not yet initialized or disabled

HAL_ADC_STATE_READY

ADC peripheral ready for use

HAL_ADC_STATE_BUSY_INTERNAL

ADC is busy to internal process (initialization, calibration)

HAL_ADC_STATE_TIMEOUT

TimeOut occurrence

HAL_ADC_STATE_ERROR_INTERNAL

Internal error occurrence

HAL_ADC_STATE_ERROR_CONFIG

Configuration error occurrence

HAL_ADC_STATE_ERROR_DMA

DMA error occurrence

HAL_ADC_STATE_REG_BUSY

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)
HAL_ADC_STATE_REG_EOC	Conversion data available on group regular
HAL_ADC_STATE_REG_OVR	Overflow occurrence
HAL_ADC_STATE_REG_EOSMP	Not available on STM32F0 device: End Of Sampling flag raised
HAL_ADC_STATE_INJ_BUSY	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 Source

ADC_EXTERNALTRIGCONV_T6_TRGO
 ADC_EXTERNALTRIGCONV_T21_CC2
 ADC_EXTERNALTRIGCONV_T2_TRGO
 ADC_EXTERNALTRIGCONV_T2_CC4
 ADC_EXTERNALTRIGCONV_T22_TRGO
 ADC_EXTERNALTRIGCONV_T3_TRGO
 ADC_EXTERNALTRIGCONV_EXT_IT11
 ADC_SOFTWARE_START
 ADC_EXTERNALTRIGCONV_T21_TRGO
 ADC_EXTERNALTRIGCONV_T2_CC3
 IS_ADC_EXTTRIG

ADC Flags Definition

ADC_FLAG_RDY	ADC Ready (ADRDY) flag
ADC_FLAG_EOSMP	ADC End of Sampling flag
ADC_FLAG_EOC	ADC End of Regular Conversion flag

ADC_FLAG_EOS	ADC End of Regular sequence of Conversions flag
ADC_FLAG_OVR	ADC overrun flag
ADC_FLAG_AWD	ADC Analog watchdog flag
ADC_FLAG_EOCAL	ADC End of Calibration flag
ADC_FLAG_ALL	

ADC Interrupts Definition

ADC_IT_RDY	ADC Ready (ADRDY) interrupt source
ADC_IT_EOSMP	ADC End of Sampling interrupt source
ADC_IT_EOC	ADC End of Regular Conversion interrupt source
ADC_IT_EOS	ADC End of Regular sequence of Conversions interrupt source
ADC_IT_OVR	ADC overrun interrupt source
ADC_IT_AWD	ADC Analog watchdog 1 interrupt source
ADC_IT_EOCAL	ADC End of Calibration interrupt source

ADC Overrun

ADC_OVR_DATA_PRESERVED	
ADC_OVR_DATA_OVERWRITTEN	

ADC Oversampling Ratio

ADC_OVERSAMPLING_RATIO_2	ADC Oversampling ratio 2x
ADC_OVERSAMPLING_RATIO_4	ADC Oversampling ratio 4x
ADC_OVERSAMPLING_RATIO_8	ADC Oversampling ratio 8x
ADC_OVERSAMPLING_RATIO_16	ADC Oversampling ratio 16x
ADC_OVERSAMPLING_RATIO_32	ADC Oversampling ratio 32x
ADC_OVERSAMPLING_RATIO_64	ADC Oversampling ratio 64x
ADC_OVERSAMPLING_RATIO_128	ADC Oversampling ratio 128x
ADC_OVERSAMPLING_RATIO_256	ADC Oversampling ratio 256x

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 External Trigger Source Edge for Regular Group

ADC_EXTERNALTRIGCONVEDGE_NONE	
ADC_EXTERNALTRIGCONVEDGE_RISING	

ADC_EXTERNALTRIGCONVEDGE_FALLING

ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

ADC Regular Nb Conversion Verification

IS_ADC_REGULAR_NB_CONV

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 Right Bit Shift

ADC_RIGHTBITSHIFT_NONE ADC No bit shift for oversampling

ADC_RIGHTBITSHIFT_1 ADC 1 bit shift for oversampling

ADC_RIGHTBITSHIFT_2 ADC 2 bits shift for oversampling

ADC_RIGHTBITSHIFT_3 ADC 3 bits shift for oversampling

ADC_RIGHTBITSHIFT_4 ADC 4 bits shift for oversampling

ADC_RIGHTBITSHIFT_5 ADC 5 bits shift for oversampling

ADC_RIGHTBITSHIFT_6 ADC 6 bits shift for oversampling

ADC_RIGHTBITSHIFT_7 ADC 7 bits shift for oversampling

ADC_RIGHTBITSHIFT_8 ADC 8 bits shift for oversampling

ADC Sampling Cycles

ADC_SAMPLETIME_1CYCLE_5 ADC sampling time 1.5 cycle

ADC_SAMPLETIME_7CYCLES_5 ADC sampling time 7.5 CYCLES

ADC_SAMPLETIME_13CYCLES_5 ADC sampling time 13.5 CYCLES

ADC_SAMPLETIME_28CYCLES_5 ADC sampling time 28.5 CYCLES

ADC_SAMPLETIME_41CYCLES_5 ADC sampling time 41.5 CYCLES

ADC_SAMPLETIME_55CYCLES_5 ADC sampling time 55.5 CYCLES

ADC_SAMPLETIME_71CYCLES_5 ADC sampling time 71.5 CYCLES

ADC_SAMPLETIME_239CYCLES_5 ADC sampling time 239.5 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

ADC SYSCFG internal paths Flags Definition

ADC_FLAG_SENSOR

ADC_FLAG_VREFINT

ADC TimeOut Values

ADC_ENABLE_TIMEOUT

ADC_DISABLE_TIMEOUT

ADC_STOP_CONVERSION_TIMEOUT

ADC_DELAY_10US_MIN_CPU_CYCLES

ADC Triggered Oversampling Mode

ADC_TRIGGEREDMODE_SINGLE_TRIGGER ADC No bit shift for oversampling

ADC_TRIGGEREDMODE_MULTI_TRIGGER ADC No bit shift for oversampling

5 HAL ADC Extension Driver

5.1 ADCEX Firmware driver API description

5.1.1 ADC specific features

1. Self calibration.

5.1.2 How to use this driver

1. Call HAL_ADCEX_Calibration_Start() to start calibration
2. Read the calibration factor using HAL_ADCEX_Calibration_GetValue()
3. User can set a his calibration factor using HAL_ADCEX_Calibration_SetValue()

5.1.3 HAL_ADCEX_Calibration_Start

Function Name	HAL_StatusTypeDef HAL_ADCEX_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)
Function Description	Start an automatic calibration.
Parameters	• hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. • SingleDiff: Selection of single-ended or differential input This parameter can be only of the following values: ADC_SINGLE_ENDED: Channel in mode input single ended
Return values	• HAL: status

5.1.4 HAL_ADCEX_Calibration_GetValue

Function Name	uint32_t HAL_ADCEX_Calibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)
Function Description	Get the calibration factor.
Parameters	• hadc: ADC handle. • SingleDiff: This parameter can be only: ADC_SINGLE_ENDED: Channel in mode input single ended.
Return values	• Calibration: value.

5.1.5 HAL_ADCEX_Calibration_SetValue

Function Name	HAL_StatusTypeDef HAL_ADCEX_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)
Function Description	Set the calibration factor to overwrite automatic conversion result.
Parameters	• hadc: ADC handle • SingleDiff: This parameter can be only: ADC_SINGLE_ENDED: Channel in mode input single ended. • CalibrationFactor: Calibration factor (coded on 7 bits)

maximum)

Return values

- **HAL:** state

5.1.6 HAL_ADCEx_EnableVREFINT

Function Name **HAL_StatusTypeDef HAL_ADCEx_EnableVREFINT (void)**

Function Description Enables the buffer of Vrefint for the ADC, required when device is in mode low-power (low-power run, low-power sleep or stop mode) This function must be called before function HAL_ADC_Init() (in case of previous ADC operations: function HAL_ADC_DeInit() must be called first) For more details on procedure and buffer current consumption, refer to device reference manual.

Return values

- **None:**

Notes

- This is functional only if the LOCK is not set.

5.1.7 HAL_ADCEx_DisableVREFINT

Function Name **void HAL_ADCEx_DisableVREFINT (void)**

Function Description Disables the Buffer Vrefint for the ADC.

Return values

- **None:**

Notes

- This is functional only if the LOCK is not set.

5.1.8 HAL_ADCEx_EnableVREFINTTempSensor

Function Name **HAL_StatusTypeDef
HAL_ADCEx_EnableVREFINTTempSensor (void)**

Function Description Enables the buffer of temperature sensor for the ADC, required when device is in mode low-power (low-power run, low-power sleep or stop mode) This function must be called before function HAL_ADC_Init() (in case of previous ADC operations: function HAL_ADC_DeInit() must be called first) For more details on procedure and buffer current consumption, refer to device reference manual.

Return values

- **None:**

Notes

- This is functional only if the LOCK is not set.

5.1.9 HAL_ADCEx_DisableVREFINTTempSensor

Function Name **void HAL_ADCEx_DisableVREFINTTempSensor (void)**

Function Description Disables the VEREFINT and Sensor for the ADC.

Return values

- **None:**

Notes

- This is functional only if the LOCK is not set.

5.2 ADCEX Firmware driver defines

5.2.1 ADCEX

ADC Calibration Factor Length Verification

IS_ADC_CALFACT

Description:

- Calibration factor length verification (7 bits maximum)

Parameters:

- `_Calibration_Factor_`: Calibration factor value

Return value:

- None

ADC Single Ended

ADC_SINGLE_ENDED

6 HAL COMP Generic Driver

6.1 COMP Firmware driver registers structures

6.1.1 COMP_InitTypeDef

Data Fields

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

Field Documentation

- *uint32_t COMP_InitTypeDef::InvertingInput*
Selects the inverting input of the comparator. This parameter can be a value of [COMP_InvertingInput](#)
- *uint32_t COMP_InitTypeDef::NonInvertingInput*
Selects the non inverting input of the comparator. This parameter can be a value of [COMP_NonInvertingInput](#)
- *uint32_t COMP_InitTypeDef::LPTIMConnection*
Selects if the COMP connection to the LPTIM is established or not. This parameter can be a value of [COMP_LPTIMConnection](#)
- *uint32_t COMP_InitTypeDef::OutputPol*
Selects the output polarity of the comparator. This parameter can be a value of [COMP_OutputPolarity](#)
- *uint32_t COMP_InitTypeDef::Mode*
Selects the operating consumption mode of the comparator to adjust the speed/consumption. This parameter can be a value of [COMP_Mode](#)
- *uint32_t COMP_InitTypeDef::WindowMode*
Selects the window mode of the comparator. This parameter can be a value of [COMP_WindowMode](#)
- *uint32_t COMP_InitTypeDef::TriggerMode*
Selects the trigger mode of the comparator (interrupt mode). This parameter can be a value of [COMP_TriggerMode](#)

6.1.2 COMP_HandleTypeDef

Data Fields

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

Field Documentation

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

6.2 COMP Firmware driver API description

6.2.1 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:

- [HAL_COMP_Init\(\)](#)
- [HAL_COMP_DeInit\(\)](#)
- [HAL_COMP_MspInit\(\)](#)
- [HAL_COMP_MspDeInit\(\)](#)

6.2.2 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\(\)](#)

6.2.3 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\(\)](#)

6.2.4 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\(\)](#)

6.2.5 HAL_COMP_Init

Function Name **HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef**

*** hcomp)**

Function Description	Initializes the COMP according to the specified parameters in the COMP_InitTypeDef and create the associated handle.
Parameters	• hcomp: COMP handle
Return values	• HAL: status
Notes	• If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset. • When the LPTIM connection is enabled, the following pins LPTIM_IN1(PB5, PC0) and LPTIM_IN2(PB7, PC2) should not be configured in AF.

6.2.6 HAL_COMP_DeInit

Function Name	HAL_StatusTypeDef HAL_COMP_DeInit (COMP_HandleTypeDef * hcomp)
Function Description	DeInitializes the COMP peripheral.
Parameters	• hcomp: COMP handle
Return values	• HAL: status
Notes	• Deinitialization can't be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

6.2.7 HAL_COMP_MspInit

Function Name	void HAL_COMP_MspInit (COMP_HandleTypeDef * hcomp)
Function Description	Initializes the COMP MSP.
Parameters	• hcomp: COMP handle
Return values	• None:

6.2.8 HAL_COMP_MspDeInit

Function Name	void HAL_COMP_MspDeInit (COMP_HandleTypeDef * hcomp)
Function Description	DeInitializes COMP MSP.
Parameters	• hcomp: COMP handle
Return values	• None:

6.2.9 HAL_COMP_Start

Function Name	HAL_StatusTypeDef HAL_COMP_Start (COMP_HandleTypeDef * hcomp)
Function Description	Start the comparator.
Parameters	• hcomp: COMP handle
Return values	• HAL: status

6.2.10 HAL_COMP_Stop

Function Name	HAL_StatusTypeDef HAL_COMP_Stop (COMP_HandleTypeDef * hcomp)
Function Description	Stop the comparator.
Parameters	• hcomp: COMP handle
Return values	• HAL: status

6.2.11 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	• hcomp: COMP handle
Return values	• HAL: status.

6.2.12 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	• hcomp: COMP handle
Return values	• HAL: status

6.2.13 HAL_COMP_IRQHandler

Function Name	void HAL_COMP_IRQHandler (COMP_HandleTypeDef * hcomp)
Function Description	Comparator IRQ Handler.
Parameters	• hcomp: COMP handle
Return values	• HAL: status

6.2.14 HAL_COMP_Lock

Function Name	HAL_StatusTypeDef HAL_COMP_Lock (COMP_HandleTypeDef * hcomp)
Function Description	Lock the selected comparator configuration.
Parameters	• hcomp: COMP handle
Return values	• HAL: status

6.2.15 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.

6.2.16 HAL_COMP_TriggerCallback

Function Name	void HAL_COMP_TriggerCallback (COMP_HandleTypeDef * hcomp)
Function Description	Comparator callback.
Parameters	hcomp: COMP handle
Return values	None:

6.2.17 HAL_COMP_GetState

Function Name	HAL_COMP_StateTypeDef HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)
Function Description	Return the COMP state.
Parameters	hcomp: : COMP handle
Return values	HAL: state

6.3 COMP Firmware driver defines**6.3.1 COMP****COMP Exported Constants**

COMP_CSR_UPDATE_PARAMETERS_MASK

COMP_LOCK_DISABLE

COMP_LOCK_ENABLE

COMP_STATE_BIT_LOCK

COMP Exported Macros

__HAL_COMP_RESET_HANDLE_STATE

Description:

- Reset the state machine associated to the handler.

Parameters:

- __HANDLE__:** COMP handle.

Return value:

- None.

Description:

- Enables the specified comparator.

Parameters:

- __HANDLE__:** COMP handle.

Return value:

- None.

__HAL_COMP_ENABLE

`__HAL_COMP_DISABLE`**Description:**

- Disables the specified comparator.

Parameters:

- `__HANDLE__`: COMP handle.

Return value:

- None.

`__HAL_COMP_LOCK`**Description:**

- Lock the specified comparator configuration.

Parameters:

- `__HANDLE__`: COMP handle.

Return value:

- None.

`__HAL_COMP_GET_FLAG`**Description:**

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

Parameters:

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

Return value:

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

`__HAL_COMP_COMP1_EXTI_ENABLE_RISING_EDGE``__HAL_COMP_COMP2_EXTI_ENABLE_RISING_EDGE``__HAL_COMP_COMP1_EXTI_DISABLE_RISING_EDGE``__HAL_COMP_COMP2_EXTI_DISABLE_RISING_EDGE``__HAL_COMP_COMP1_EXTI_ENABLE_FALLING_EDGE`**Description:**

- Enable the Exti Line falling edge trigger.

__HAL_COMP_COMP2_EXTI_ENABLE_FALLING_EDGE

__HAL_COMP_COMP1_EXTI_DISABLE_FALLING_EDGE

__HAL_COMP_COMP2_EXTI_DISABLE_FALLING_EDGE

__HAL_COMP_COMP1_EXTI_ENABLE_RISING_FALLING_EDGE

__HAL_COMP_COMP1_EXTI_DISABLE_RISING_FALLING_EDGE

__HAL_COMP_COMP2_EXTI_ENABLE_RISING_FALLING_EDGE

__HAL_COMP_COMP2_EXTI_DISABLE_RISING_FALLING_EDGE

COMP_GET_EXTI_LINE

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 COMP2 EXTI line rising & falling edge trigger.

Return value:

- None.

Description:

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

Return value:

- None.

Description:

- Get the specified EXTI line for a comparator instance.

Parameters:

- `__INSTANCE__`: specifies the COMP instance.

Return value:

__HAL_COMP_COMP1_EXTI_ENABLE_IT

- value: of

Description:

- Enable the COMP Exti Line.

Return value:

- None.

__HAL_COMP_COMP2_EXTI_ENABLE_IT

__HAL_COMP_COMP1_EXTI_DISABLE_IT

Description:

- Disable the COMP Exti Line.

Return value:

- None.

__HAL_COMP_COMP2_EXTI_DISABLE_IT

__HAL_COMP_COMP1_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP EXTI line.

Return value:

- None

__HAL_COMP_COMP2_EXTI_GENERATE_SWIT

__HAL_COMP_COMP1_EXTI_ENABLE_EVENT

Description:

- Enable the COMP EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_EVENT

__HAL_COMP_COMP1_EXTI_DISABLE_EVENT

Description:

- Disable the COMP EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_EVENT

__HAL_COMP_COMP1_EXTI_GET_FLAG

Description:

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

Return value:

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

__HAL_COMP_COMP2_EXTI_GET_FLAG
__HAL_COMP_COMP1_EXTI_CLEAR_FLAG

Description:

- Clear the COMP Exti flags.

Return value:

- None.

__HAL_COMP_COMP2_EXTI_CLEAR_FLAG

COMP EXTI line definition

COMP_EXTI_LINE_COMP2 External interrupt line 22 Connected to COMP2

COMP_EXTI_LINE_COMP1 External interrupt line 21 Connected to COMP1

COMP inverting input definitions

COMP_INVERTINGINPUT_VREFINT VREFINT connected to comparator1 inverting input

COMP_INVERTINGINPUT_IO1 I/O1 connected to comparator inverting input (PA0) for COMP1 and (PA2) for COMP2

COMP_INVERTINGINPUT_DAC1 DAC1_OUT (PA4) connected to comparator inverting input

COMP_INVERTINGINPUT_IO2 I/O2 (PA5) connected to comparator inverting input

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_IO3 I/O3 (PB3) for COMP2 connected to comparator inverting input

IS_COMP_INVERTINGINPUT

COMP Low power timer connection definition

COMP_LPTIMCONNECTION_DISABLED COMPx signal is gated

COMP_LPTIMCONNECTION_IN1_ENABLED COMPx signal is connected to LPTIM input 1

COMP_LPTIMCONNECTION_IN2_ENABLED COMPx signal is connected to LPTIM input 2

IS_COMP1_LPTIMCONNECTION

IS_COMP2_LPTIMCONNECTION

IS_COMP2_LPTIMCONNECTION_RESTRICTED

COMP mode definition

COMP_MODE_HIGHSPEED High Speed

COMP_MODE_LOWSPEED Low Speed

IS_COMP_MODE
COMP non inverting input definitions

COMP_NONINVERTINGINPUT_IO1	I/O1 (PA3) connected to comparator non inverting input
COMP_NONINVERTINGINPUT_IO2	I/O2 (PB4) connected to comparator non inverting input
COMP_NONINVERTINGINPUT_IO3	I/O3 (PB5) connected to comparator non inverting input
COMP_NONINVERTINGINPUT_IO4	I/O1 (PB6) connected to comparator non inverting input
COMP_NONINVERTINGINPUT_IO5	I/O3 (PB7) connected to comparator non inverting input
COMP_NONINVERTINGINPUT_IO6	I/O3 (PA7) connected to comparator non inverting input
COMP_NONINVERTINGINPUT_IO7	Reserved
COMP_NONINVERTINGINPUT_IO8	Reserved

IS_COMP_NONINVERTINGINPUT

COMP output level definition

COMP_OUTPUTLEVEL_LOW

COMP_OUTPUTLEVEL_HIGH

COMP output polarity definitions

COMP_OUTPUTPOL_NONINVERTED COMP output on GPIO isn't inverted

COMP_OUTPUTPOL_INVERTED COMP output on GPIO is inverted

IS_COMP_OUTPUTPOL

COMP trigger mode definition

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 window mode definition

COMP_WINDOWMODE_DISABLE	Window mode disabled (Plus input of comparator 1 connected to PA1)
COMP_WINDOWMODE_ENABLE	Window mode enabled: Plus input of comparator 1 shorted with Plus input of comparator 2
IS_COMP_WINDOWMODE	
IS_COMP_WINDOWMODE_INSTANCE	

7 HAL COMP Extension Driver

7.1 COMPEX Firmware driver API description

7.1.1 COMP peripheral Extended features

Comparing to other previous devices, the COMP interface for STM32L0XX devices contains the following additional features

- Possibility to enable or disable the VREFINT which is used as input to the comparator.

7.1.2 HAL_COMPEx_EnableVREFINT

Function Name **void HAL_COMPEx_EnableVREFINT (void)**

Function Description Enables the Buffer Vrefint for the COMP.

Return values • **None:**

Notes • This is functional only if the LOCK bit is not set

7.1.3 HAL_COMPEx_DisableVREFINT

Function Name **void HAL_COMPEx_DisableVREFINT (void)**

Function Description Disables the Buffer Vrefint for the COMP.

Return values • **None:**

Notes • This is functional only if the LOCK bit is not set

8 HAL CORTEX Generic Driver

8.1 CORTEX Firmware driver registers structures

8.1.1 MPU_Region_InitTypeDef

Data Fields

- *uint32_t* **BaseAddress**
- *uint8_t* **Enable**
- *uint8_t* **Number**
- *uint8_t* **Size**
- *uint8_t* **SubRegionDisable**
- *uint8_t* **TypeExtField**
- *uint8_t* **AccessPermission**
- *uint8_t* **DisableExec**
- *uint8_t* **IsShareable**
- *uint8_t* **IsCacheable**
- *uint8_t* **IsBufferable**

Field Documentation

- **uint32_t MPU_Region_InitTypeDef::BaseAddress**
Specifies the base address of the region to protect.
- **uint8_t MPU_Region_InitTypeDef::Enable**
Specifies the status of the region. This parameter can be a value of [CORTEX_MPU_Region_Enable](#)
- **uint8_t MPU_Region_InitTypeDef::Number**
Specifies the number of the region to protect. This parameter can be a value of [CORTEX_MPU_Region_Number](#)
- **uint8_t MPU_Region_InitTypeDef::Size**
Specifies the size of the region to protect. This parameter can be a value of [CORTEX_MPU_Region_Size](#)
- **uint8_t MPU_Region_InitTypeDef::SubRegionDisable**
Specifies the number of the subregion protection to disable. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- **uint8_t MPU_Region_InitTypeDef::TypeExtField**
This parameter is NOT used but is kept to keep API unified through all families
- **uint8_t MPU_Region_InitTypeDef::AccessPermission**
Specifies the region access permission type. This parameter can be a value of [CORTEX_MPU_Region_Permission_Attributes](#)
- **uint8_t MPU_Region_InitTypeDef::DisableExec**
Specifies the instruction access status. This parameter can be a value of [CORTEX_MPU_Instruction_Access](#)
- **uint8_t MPU_Region_InitTypeDef::IsShareable**
Specifies the shareability status of the protected region. This parameter can be a value of [CORTEX_MPU_Access_Shareable](#)
- **uint8_t MPU_Region_InitTypeDef::IsCacheable**
Specifies the cacheable status of the region protected. This parameter can be a value of [CORTEX_MPU_Access_Cacheable](#)

- **`uint8_t MPU_Region_InitTypeDef::IsBufferable`**
Specifies the bufferable status of the protected region. This parameter can be a value of **`CORTEX_MPU_Access_Bufferable`**

8.2 CORTEX Firmware driver API description

8.2.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

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

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

How to configure SysTick using CORTEX HAL driver

Setup SysTick Timer for time base

- The `HAL_SYSTICK_Config()` function calls the `SysTick_Config()` function which is a CMSIS function that:
 - Configures the SysTick Reload register with value passed as function parameter.
 - Configures the SysTick IRQ priority to the lowest value (0x03).
 - Resets the SysTick Counter register.
 - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
 - Enables the SysTick Interrupt.
 - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK_Div8 by calling the macro `__HAL_CORTEX_SYSTICKCLK_CONFIG(SYSTICK_CLKSOURCE_HCLK_DIV8)` just after the `HAL_SYSTICK_Config()` function call. The `__HAL_CORTEX_SYSTICKCLK_CONFIG()` macro is defined inside the `stm32l0xx_hal_cortex.h` file.
- You can change the SysTick IRQ priority by calling the `HAL_NVIC_SetPriority(SysTick_IRQn,...)` function just after the `HAL_SYSTICK_Config()` function call. The `HAL_NVIC_SetPriority()` call the `NVIC_SetPriority()` function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
 - Reload Value is the parameter to be passed for `HAL_SYSTICK_Config()` function
 - Reload Value should not exceed 0xFFFFFF

8.2.2 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\(\)](#)
- [HAL_MPU_Disable\(\)](#)
- [HAL_MPU_Enable\(\)](#)

8.2.3 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\(\)](#)
- [HAL_MPU_ConfigRegion\(\)](#)

8.2.4 HAL_NVIC_SetPriority

Function Name **void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)**

Function Description Sets the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number . This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file)
- **PreemptPriority:** The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 3. A lower priority value indicates a higher priority
- **SubPriority:** The subpriority level for the IRQ channel. with stm32l0xx devices, this parameter is a dummy value and it is ignored, because no subpriority supported in Cortex M0+ based products.

Return values

- **None:**

8.2.5 HAL_NVIC_EnableIRQ

Function Name **void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)**

Function Description Enables a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number . This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file)

Return values

- **None:**

Notes

- To configure interrupts priority correctly, the

NVIC_PriorityGroupConfig() function should be called before.

8.2.6 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	IRQn: External interrupt number . This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file)
Return values	None:

8.2.7 HAL_NVIC_SystemReset

Function Name	void HAL_NVIC_SystemReset (void)
Function Description	Initiates a system reset request to reset the MCU.
Return values	None:

8.2.8 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	TicksNumb: Specifies the ticks Number of ticks between two interrupts.
Return values	status: - 0 Function succeeded.1 Function failed.

8.2.9 HAL_MPU_Disable

Function Name	__STATIC_INLINE void HAL_MPU_Disable (void)
Function Description	Disable the MPU.
Return values	None:

8.2.10 HAL_MPU_Enable

Function Name	__STATIC_INLINE void HAL_MPU_Enable (uint32_t MPU_Control)
Function Description	Enable the MPU.
Parameters	MPU_Control: Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged access to the default memory This parameter can be one of the following values: MPU_HFNMI_PRIVDEF_NONEMPU_HARDFAULT_NMIMPU_PRIVILEGE D_DEFAULTMPU_HFNMI_PRIVDEF
Return	None:

values

8.2.11 HAL_NVIC_GetPriority

Function Name **uint32_t HAL_NVIC_GetPriority (IRQn_Type IRQn)**

Function Description Gets the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l0xxx.h))

Return values

- None

8.2.12 HAL_NVIC_SetPendingIRQ

Function Name **void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)**

Function Description Sets Pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file)

Return values

- **None:**

8.2.13 HAL_NVIC_GetPendingIRQ

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

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

Parameters

- **IRQn:** External interrupt number . This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file)

Return values

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

8.2.14 HAL_NVIC_ClearPendingIRQ

Function Name **void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)**

Function Description Clears the pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number . This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32l0xx.h file)

Return values

- **None:**

8.2.15 HAL_SYSTICK_CLKSourceConfig

Function Name **void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)**

Function Description	Configures the SysTick clock source.
Parameters	• CLKSource: specifies the SysTick clock source. This parameter can be one of the following values: SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source. SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.
Return values	• None:

8.2.16 HAL_SYSTICK_IRQHandler

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

8.2.17 HAL_SYSTICK_Callback

Function Name	void HAL_SYSTICK_Callback (void)
Function Description	SYSTICK callback.
Return values	• None:

8.2.18 HAL_MPU_ConfigRegion

Function Name	void HAL_MPU_ConfigRegion (MPU_Region_InitTypeDef * MPU_Init)
Function Description	Initialize and configure the Region and the memory to be protected.
Parameters	• MPU_Init: Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.
Return values	• None:

8.3 CORTEX Firmware driver defines

8.3.1 CORTEX

CORTEX Exported Constants

IS_NVIC_PREEMPTION_PRIORITY

IS_NVIC_DEVICE_IRQ

CORTEX Exported Macros

__HAL_CORTEX_SYSTICKCLK_CONFIG

Description:

- Configures the SysTick clock source.

Parameters:

- **__CLKSRC__**: specifies the SysTick clock source. This parameter can be one of the following values:
 - SYSTICK_CLKSOURCE_HCLK_DIV8:

AHB clock divided by 8 selected as SysTick clock source.

- SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.

Return value:

- None

CORTEX MPU Instruction Access Bufferable

MPU_ACCESS_BUFFERABLE

MPU_ACCESS_NOT_BUFFERABLE

CORTEX MPU Instruction Access Cacheable

MPU_ACCESS_CACHEABLE

MPU_ACCESS_NOT_CACHEABLE

CORTEX MPU Instruction Access Shareable

MPU_ACCESS_SHAREABLE

MPU_ACCESS_NOT_SHAREABLE

CORTEX MPU HFNMI and PRIVILEGED Access control

MPU_HFNMI_PRIVDEF_NONE

MPU_HARDFFAULT_NMI

MPU_PRIVILEGED_DEFAULT

MPU_HFNMI_PRIVDEF

CORTEX MPU Instruction Access

MPU_INSTRUCTION_ACCESS_ENABLE

MPU_INSTRUCTION_ACCESS_DISABLE

CORTEX MPU Region Enable

MPU_REGION_ENABLE

MPU_REGION_DISABLE

CORTEX MPU Region Number

MPU_REGION_NUMBER0

MPU_REGION_NUMBER1

MPU_REGION_NUMBER2

MPU_REGION_NUMBER3

MPU_REGION_NUMBER4

MPU_REGION_NUMBER5

MPU_REGION_NUMBER6

MPU_REGION_NUMBER7

CORTEX MPU Region Permission Attributes

MPU_REGION_NO_ACCESS

MPU_REGION_PRIV_RW
MPU_REGION_PRIV_RW_URO
MPU_REGION_FULL_ACCESS
MPU_REGION_PRIV_RO
MPU_REGION_PRIV_RO_URO

CORTEX MPU Region Size

MPU_REGION_SIZE_32B
MPU_REGION_SIZE_64B
MPU_REGION_SIZE_128B
MPU_REGION_SIZE_256B
MPU_REGION_SIZE_512B
MPU_REGION_SIZE_1KB
MPU_REGION_SIZE_2KB
MPU_REGION_SIZE_4KB
MPU_REGION_SIZE_8KB
MPU_REGION_SIZE_16KB
MPU_REGION_SIZE_32KB
MPU_REGION_SIZE_64KB
MPU_REGION_SIZE_128KB
MPU_REGION_SIZE_256KB
MPU_REGION_SIZE_512KB
MPU_REGION_SIZE_1MB
MPU_REGION_SIZE_2MB
MPU_REGION_SIZE_4MB
MPU_REGION_SIZE_8MB
MPU_REGION_SIZE_16MB
MPU_REGION_SIZE_32MB
MPU_REGION_SIZE_64MB
MPU_REGION_SIZE_128MB
MPU_REGION_SIZE_256MB
MPU_REGION_SIZE_512MB
MPU_REGION_SIZE_1GB
MPU_REGION_SIZE_2GB
MPU_REGION_SIZE_4GB

CORTEX SysTick Clock Source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

IS_SYSTICK_CLK_SOURCE

9 HAL CRC Generic Driver

9.1 CRC Firmware driver registers structures

9.1.1 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 [CRC_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

9.1.2 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

9.2 CRC Firmware driver API description

9.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

1. Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle
2. DeInitialize the CRC peripheral
3. Initialize the CRC MSP
4. DeInitialize CRC MSP

This section contains the following APIs:

- [HAL_CRC_Init\(\)](#)
- [HAL_CRC_DeInit\(\)](#)
- [HAL_CRC_MspInit\(\)](#)
- [HAL_CRC_MspDeInit\(\)](#)

9.2.2 Peripheral Control functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [HAL_CRC_Accumulate\(\)](#)
- [HAL_CRC_Calculate\(\)](#)

9.2.3 Peripheral State functions

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

This section contains the following APIs:

- [HAL_CRC_GetState\(\)](#)

9.2.4 HAL_CRC_Init

Function Name	HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)
Function Description	Initializes the CRC according to the specified parameters in the CRC_InitTypeDef and creates the associated handle.
Parameters	• hcrc: CRC handle
Return values	• HAL: status

9.2.5 HAL_CRC_DeInit

Function Name	HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)
Function Description	DeInitializes the CRC peripheral.
Parameters	• hcrc: CRC handle
Return values	• HAL: status

9.2.6 HAL_CRC_MspInit

Function Name	void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)
Function Description	Initializes the CRC MSP.
Parameters	• hcrc: CRC handle
Return values	• None:

9.2.7 HAL_CRC_MspDeInit

Function Name	void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)
Function Description	DeInitializes the CRC MSP.
Parameters	• hcrc: CRC handle
Return values	• None:

9.2.8 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	• hcrc: CRC handle • pBuffer: pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat. • BufferLength: input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).
Return values	• uint32_t: CRC (returned value LSBs for CRC shorter than 32 bits)
Notes	• By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

9.2.9 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	• hcrc: CRC handle • pBuffer: pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat. • BufferLength: input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).
Return values	• uint32_t: CRC (returned value LSBs for CRC shorter than 32 bits)
Notes	• By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

9.2.10 HAL_CRC_GetState

Function Name	HAL_CRC_StateTypeDef HAL_CRC_GetState (CRC_HandleTypeDef * hcrc)
Function Description	Returns the CRC state.
Parameters	• hcrc: CRC handle
Return values	• HAL: state

9.3 CRC Firmware driver defines

9.3.1 CRC

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

Indicates whether or not default polynomial is used

DEFAULT_POLYNOMIAL_ENABLE

DEFAULT_POLYNOMIAL_DISABLE

Default CRC generating polynomial

DEFAULT_CRC32_POLY

CRC Exported Constants

HAL_CRC_Input_Data_Reverse

HAL_CRC_Output_Data_Reverse

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.

__HAL_CRC_SET_IDR

Parameters:

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

Return value:

- None.

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

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

__HAL_CRC_GET_IDR

CRC input buffer format

CRC_INPUTDATA_FORMAT_UNDEFINED

CRC_INPUTDATA_FORMAT_BYTES

CRC_INPUTDATA_FORMAT_HALFWORDS

CRC_INPUTDATA_FORMAT_WORDS

Polynomial sizes to configure the IP

CRC_POLYLENGTH_32B

CRC_POLYLENGTH_16B

CRC_POLYLENGTH_8B

CRC_POLYLENGTH_7B

CRC polynomial possible sizes actual definitions

HAL_CRC_LENGTH_32B

HAL_CRC_LENGTH_16B

HAL_CRC_LENGTH_8B

HAL_CRC_LENGTH_7B

10 HAL CRC Extension Driver

10.1 CRCEX Firmware driver API description

10.1.1 CRC Extended features functions

This subsection provides function allowing to:

- Set CRC polynomial if different from default one.

This section contains the following APIs:

- [HAL_CRCEX_Polynomial_Set\(\)](#)
- [HAL_CRCEX_Input_Data_Reverse\(\)](#)
- [HAL_CRCEX_Output_Data_Reverse\(\)](#)

10.1.2 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	• 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	• HAL: status

10.1.3 HAL_CRCEX_Input_Data_Reverse

Function Name	HAL_StatusTypeDef HAL_CRCEX_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)
Function Description	Set the Reverse Input data mode.
Parameters	• hcrc: CRC handle • InputReverseMode: Input Data inversion mode This parameter can be one of the following values: CRC_INPUTDATA_INVERSION_NONE: no change in bit order (default value) CRC_INPUTDATA_INVERSION_BYTE: Byte-wise bit reversal CRC_INPUTDATA_INVERSION_HALFWORD: HalfWord-wise bit

reversalCRC_INPUTDATA_INVERSION_WORD: Word-wise
bit reversal

Return values

- **HAL:** status

10.1.4 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

- **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

- **HAL:** status

10.2 CRCEX Firmware driver defines

10.2.1 CRCEX

CRCEX 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.

CRC Extended input data inversion modes

CRC_INPUTDATA_INVERSION_NONE

CRC_INPUTDATA_INVERSION_BYTE

CRC_INPUTDATA_INVERSION_HALFWORD

CRC_INPUTDATA_INVERSION_WORD

CRC Extended output data inversion modes

CRC_OUTPUTDATA_INVERSION_DISABLE

CRC_OUTPUTDATA_INVERSION_ENABLE

11 HAL CRYPT Generic Driver

11.1 CRYPT Firmware driver registers structures

11.1.1 CRYPT_InitTypeDef

Data Fields

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

Field Documentation

- *uint32_t* **CRYPT_InitTypeDef::DataType**
32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of [CRYPT_Data_Type](#)
- *uint8_t ** **CRYPT_InitTypeDef::pKey**
The key used for encryption/decryption
- *uint8_t ** **CRYPT_InitTypeDef::pInitVect**
The initialization vector used also as initialization counter in CTR mode

11.1.2 CRYPT_HandleTypeDef

Data Fields

- *AES_TypeDef ** **Instance**
- *CRYPT_InitTypeDef* **Init**
- *uint8_t ** **pCrypInBuffPtr**
- *uint8_t ** **pCrypOutBuffPtr**
- *__IO uint16_t* **CrypInCount**
- *__IO uint16_t* **CrypOutCount**
- *HAL_StatusTypeDef* **Status**
- *HAL_PhaseTypeDef* **Phase**
- *DMA_HandleTypeDef ** **hdmain**
- *DMA_HandleTypeDef ** **hdmaout**
- *HAL_LockTypeDef* **Lock**
- *__IO HAL_CRYPT_STATETTypeDef* **State**

Field Documentation

- *AES_TypeDef ** **CRYPT_HandleTypeDef::Instance**
Register base address
- *CRYPT_InitTypeDef* **CRYPT_HandleTypeDef::Init**
CRYPT required parameters
- *uint8_t ** **CRYPT_HandleTypeDef::pCrypInBuffPtr**
Pointer to CRYPT processing (encryption, decryption,...) buffer

- ***uint8_t* CRYPT_HandleTypeDef::pCrypOutBuffPtr***
Pointer to CRYPT processing (encryption, decryption,...) buffer
- ***__IO uint16_t CRYPT_HandleTypeDef::CrypInCount***
Counter of inputted data
- ***__IO uint16_t CRYPT_HandleTypeDef::CrypOutCount***
Counter of outputted data
- ***HAL_StatusTypeDef CRYPT_HandleTypeDef::Status***
CRYPT peripheral status
- ***HAL_PhaseTypeDef CRYPT_HandleTypeDef::Phase***
CRYPT peripheral phase
- ***DMA_HandleTypeDef* CRYPT_HandleTypeDef::hdmain***
CRYPT In DMA handle parameters
- ***DMA_HandleTypeDef* CRYPT_HandleTypeDef::hdmaout***
CRYPT Out DMA handle parameters
- ***HAL_LockTypeDef CRYPT_HandleTypeDef::Lock***
CRYPT locking object
- ***__IO HAL_CRYPT_STATTypeDef CRYPT_HandleTypeDef::State***
CRYPT peripheral state

11.2 CRYPT Firmware driver API description

11.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRYPT according to the specified parameters in the CRYPT_InitTypeDef and creates the associated handle
- DeInitialize the CRYPT peripheral
- Initialize the CRYPT MSP
- DeInitialize CRYPT MSP

This section contains the following APIs:

- [*HAL_CRYPT_Init\(\)*](#)
- [*HAL_CRYPT_DeInit\(\)*](#)
- [*HAL_CRYPT_MspInit\(\)*](#)
- [*HAL_CRYPT_MspDeInit\(\)*](#)

11.2.2 AES processing functions

This section provides functions allowing to:

- Encrypt plaintext using AES algorithm in different chaining modes
- Decrypt cyphertext using AES algorithm in different chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode

This section contains the following APIs:

- [*HAL_CRYPT_AESECBC_Encrypt\(\)*](#)
- [*HAL_CRYPT_AESECBC_Decrypt\(\)*](#)
- [*HAL_CRYPT_AESCTR_Encrypt\(\)*](#)
- [*HAL_CRYPT_AESECBC_Decrypt\(\)*](#)

- [HAL_Cryp_AESCBC_Decrypt\(\)](#)
- [HAL_Cryp_AESCTR_Decrypt\(\)](#)
- [HAL_Cryp_AESECB_Encrypt_IT\(\)](#)
- [HAL_Cryp_AESCBC_Encrypt_IT\(\)](#)
- [HAL_Cryp_AESCTR_Encrypt_IT\(\)](#)
- [HAL_Cryp_AESECB_Decrypt_IT\(\)](#)
- [HAL_Cryp_AESCBC_Decrypt_IT\(\)](#)
- [HAL_Cryp_AESCTR_Decrypt_IT\(\)](#)
- [HAL_Cryp_AESECB_Encrypt_DMA\(\)](#)
- [HAL_Cryp_AESCBC_Encrypt_DMA\(\)](#)
- [HAL_Cryp_AESCTR_Encrypt_DMA\(\)](#)
- [HAL_Cryp_AESECB_Decrypt_DMA\(\)](#)
- [HAL_Cryp_AESCBC_Decrypt_DMA\(\)](#)
- [HAL_Cryp_AESCTR_Decrypt_DMA\(\)](#)

11.2.3 CRYPT IRQ handler management

This section provides CRYPT IRQ handler function.

This section contains the following APIs:

- [HAL_Cryp_IRQHandler\(\)](#)

11.2.4 Peripheral State functions

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

This section contains the following APIs:

- [HAL_Cryp_GetState\(\)](#)

11.2.5 DMA callback functions

This section provides DMA callback functions:

- DMA Input data transfer complete
- DMA Output data transfer complete
- DMA error

This section contains the following APIs:

- [HAL_Cryp_ErrorCallback\(\)](#)
- [HAL_Cryp_InCpltCallback\(\)](#)
- [HAL_Cryp_OutCpltCallback\(\)](#)

11.2.6 HAL_Cryp_Init

Function Name	HAL_StatusTypeDef HAL_Cryp_Init (Cryp_HandleTypeDef * hcrp)
Function Description	Initializes the CRYPT according to the specified parameters in the CRYPT_InitTypeDef and creates the associated handle.
Parameters	• hcrp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	• HAL: status

11.2.7 HAL_Cryp_DeInit

Function Name	HAL_StatusTypeDef HAL_CRYPT_DeInit (CRYPT_HandleTypeDef * hcrypt)
Function Description	DeInitializes the CRYPT peripheral.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	• HAL: status

11.2.8 HAL_CRYPT_MspInit

Function Name	void HAL_CRYPT_MspInit (CRYPT_HandleTypeDef * hcrypt)
Function Description	Initializes the CRYPT MSP.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	• None:

11.2.9 HAL_CRYPT_MspDeInit

Function Name	void HAL_CRYPT_MspDeInit (CRYPT_HandleTypeDef * hcrypt)
Function Description	DeInitializes CRYPT MSP.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	• None:

11.2.10 HAL_CRYPT_AESECB_Encrypt

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESECB_Encrypt (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES ECB encryption mode then encrypt pPlainData.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pPlainData: Pointer to the plaintext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16. • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Timeout: Specify Timeout value
Return values	• HAL: status

11.2.11 HAL_CRYPT_AESCBC_Encrypt

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESCBC_Encrypt (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES CBC encryption mode then encrypt pPlainData.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that

- contains the configuration information for CRYPT module
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16.
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
 - **Timeout:** Specify Timeout value
- Return values
- **HAL:** status

11.2.12 HAL_CRYPT_AESCTR_Encrypt

- Function Name **HAL_StatusTypeDef HAL_CRYPT_AESCTR_Encrypt (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)**
- Function Description **Initializes the CRYPT peripheral in AES CTR encryption mode then encrypt pPlainData.**
- Parameters
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16.
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
 - **Timeout:** Specify Timeout value
- Return values
- **HAL:** status

11.2.13 HAL_CRYPT_AESECBC_Decrypt

- Function Name **HAL_StatusTypeDef HAL_CRYPT_AESECBC_Decrypt (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
- Function Description **Initializes the CRYPT peripheral in AES ECB decryption mode then decrypted pCypherData.**
- Parameters
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16.
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
 - **Timeout:** Specify Timeout value
- Return values
- **HAL:** status

11.2.14 HAL_CRYPT_AESCBC_Decrypt

- Function Name **HAL_StatusTypeDef HAL_CRYPT_AESCBC_Decrypt (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)**
- Function Description **Initializes the CRYPT peripheral in AES ECB decryption mode then decrypted pCypherData.**
- Parameters
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module

- **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
- **Size:** Length of the plaintext buffer, must be a multiple of 16.
- **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

11.2.15 HAL_CRYP_AESCTR_Decrypt

Function Name

HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt
(Cryp_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)

Function Description

Initializes the CRYP peripheral in AES CTR decryption mode then decrypted pCypherData.

Parameters

- **hcryp:** pointer to a Cryp_HandleTypeDef structure that contains the configuration information for CRYP module
- **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
- **Size:** Length of the plaintext buffer, must be a multiple of 16.
- **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

11.2.16 HAL_CRYP_AESECBC_Encrypt_IT

Function Name

HAL_StatusTypeDef HAL_CRYP_AESECBC_Encrypt_IT
(Cryp_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)

Function Description

Initializes the CRYP peripheral in AES ECB encryption mode using Interrupt.

Parameters

- **hcryp:** pointer to a Cryp_HandleTypeDef structure that contains the configuration information for CRYP module
- **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
- **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes
- **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)

Return values

- **HAL:** status

11.2.17 HAL_CRYP_AESCBC_Encrypt_IT

Function Name

HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_IT
(Cryp_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)

Function Description

Initializes the CRYP peripheral in AES CBC encryption mode using Interrupt.

Parameters

- **hcryp:** pointer to a Cryp_HandleTypeDef structure that contains the configuration information for CRYP module
- **pPlainData:** Pointer to the plaintext buffer (aligned on u32)

- **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
- Return values
- **HAL:** status

11.2.18 HAL_CRYP_AESCTR_Encrypt_IT

Function Name	HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_IT (CRYP_HandleTypeDef * hcrp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode using Interrupt.
Parameters	• hcrp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pPlainData: Pointer to the plaintext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16 bytes • pCypherData: Pointer to the cyphertext buffer (aligned on u32)
Return values	• HAL: status

11.2.19 HAL_CRYP_AESECBC_Decrypt_IT

Function Name	HAL_StatusTypeDef HAL_CRYP_AESECBC_Decrypt_IT (CRYP_HandleTypeDef * hcrp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode using Interrupt.
Parameters	• hcrp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16. • pPlainData: Pointer to the plaintext buffer (aligned on u32)
Return values	• HAL: status

11.2.20 HAL_CRYP_AESCBC_Decrypt_IT

Function Name	HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_IT (CRYP_HandleTypeDef * hcrp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYP peripheral in AES CBC decryption mode using IT.
Parameters	• hcrp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16

- **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
- **HAL:** status

11.2.21 HAL_CRYPT_AESCTR_Decrypt_IT

- Function Name** HAL_StatusTypeDef HAL_CRYPT_AESCTR_Decrypt_IT (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
- Function Description** Initializes the CRYPT peripheral in AES CTR decryption mode using Interrupt.
- Parameters**
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
- Return values**
- **HAL:** status

11.2.22 HAL_CRYPT_AESECBC_Encrypt_DMA

- Function Name** HAL_StatusTypeDef HAL_CRYPT_AESECBC_Encrypt_DMA (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
- Function Description** Initializes the CRYPT peripheral in AES ECB encryption mode using DMA.
- Parameters**
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
- Return values**
- **HAL:** status

11.2.23 HAL_CRYPT_AESCBC_Encrypt_DMA

- Function Name** HAL_StatusTypeDef HAL_CRYPT_AESCBC_Encrypt_DMA (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
- Function Description** Initializes the CRYPT peripheral in AES CBC encryption mode using DMA.
- Parameters**
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16.
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
- Return values**
- **HAL:** status

11.2.24 HAL_CRYPT_AESCTR_Encrypt_DMA

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESCTR_Encrypt_DMA (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in AES CTR encryption mode using DMA.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pPlainData: Pointer to the plaintext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16. • pCypherData: Pointer to the cyphertext buffer (aligned on u32)
Return values	• HAL: status

11.2.25 HAL_CRYPT_AESECB_Decrypt_DMA

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESECB_Decrypt_DMA (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in AES ECB decryption mode using DMA.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16 bytes • pPlainData: Pointer to the plaintext buffer (aligned on u32)
Return values	• HAL: status

11.2.26 HAL_CRYPT_AESCBC_Decrypt_DMA

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESCBC_Decrypt_DMA (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in AES CBC encryption mode using DMA.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16 bytes • pPlainData: Pointer to the plaintext buffer (aligned on u32)
Return values	• HAL: status

11.2.27 HAL_CRYPT_AESCTR_Decrypt_DMA

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESCTR_Decrypt_DMA (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in AES CTR decryption mode using DMA.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16 • pPlainData: Pointer to the plaintext buffer (aligned on u32)
Return values	• HAL: status

11.2.28 HAL_CRYPT_IRQHandler

Function Name	void HAL_CRYPT_IRQHandler (CRYPT_HandleTypeDef * hcrypt)
Function Description	This function handles CRYPT interrupt request.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	• None:

11.2.29 HAL_CRYPT_GetState

Function Name	HAL_CRYPT_STATTypeDef HAL_CRYPT_GetState (CRYPT_HandleTypeDef * hcrypt)
Function Description	Returns the CRYPT state.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	• HAL: state

11.2.30 HAL_CRYPT_ErrorCallback

Function Name	void HAL_CRYPT_ErrorCallback (CRYPT_HandleTypeDef * hcrypt)
Function Description	CRYPT error callback.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	• None:

11.2.31 HAL_CRYPT_InCpltCallback

Function Name	void HAL_CRYPT_InCpltCallback (CRYPT_HandleTypeDef * hcrypt)
Function Description	Input transfer completed callback.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module

Return values

- **None:**

11.2.32 HAL_CRYPT_OutCpltCallback

Function Name **void HAL_CRYPT_OutCpltCallback (CRYPT_HandleTypeDef * hcrypt)**

Function Description Output transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module

Return values

- **None:**

11.3 CRYPT Firmware driver defines

11.3.1 CRYPT

AES Clear Flags

CRYPT_CLEARFLAG_CCF Computation Complete Flag Clear

CRYPT_CLEARFLAG_RDERR Read Error Clear

CRYPT_CLEARFLAG_WRERR Write Error Clear

AES Flags

CRYPT_FLAG_CCF Computation Complete Flag

CRYPT_FLAG_RDERR Read Error Flag

CRYPT_FLAG_WRERR Write Error Flag

AES Interrupts

CRYPT_IT_CC Computation Complete interrupt

CRYPT_IT_ERR Error interrupt

CRYPT Algo Mode Direction

CRYPT_CR_ALGOMODE_DIRECTION

CRYPT_CR_ALGOMODE_AES_ECB_ENCRYPT

CRYPT_CR_ALGOMODE_AES_ECB_KEYDERDECRYPT

CRYPT_CR_ALGOMODE_AES_CBC_ENCRYPT

CRYPT_CR_ALGOMODE_AES_CBC_KEYDERDECRYPT

CRYPT_CR_ALGOMODE_AES_CTR_ENCRYPT

CRYPT_CR_ALGOMODE_AES_CTR_DECRYPT

CRYPT Data Type

CRYPT_DATATYPE_32B

CRYPT_DATATYPE_16B

CRYPT_DATATYPE_8B

CRYPT_DATATYPE_1B

IS_CRYPT_DATATYPE

CRYP Exported Macros

<code>__HAL_CRYP_RESET_HANDLE_STATE</code>	Description: - Reset CRYP handle state. Parameters: <code>__HANDLE__</code>: specifies the CRYP handle. Return value: - None
<code>__HAL_CRYP_ENABLE</code>	Description: - Enable/Disable the CRYP peripheral. Parameters: <code>__HANDLE__</code>: specifies the CRYP handle. Return value: - None
<code>__HAL_CRYP_DISABLE</code>	
<code>__HAL_CRYP_SET_MODE</code>	Description: - Set the algorithm mode: AES-ECB, AES-CBC, AES-CTR, DES-ECB, DES-CBC,... Parameters: <code>__HANDLE__</code>: specifies the CRYP handle. <code>__MODE__</code>: The algorithm mode. Return value: - None
<code>__HAL_CRYP_GET_FLAG</code>	Description: - Check whether the specified CRYP flag is set or not. Parameters: <code>__HANDLE__</code>: specifies the CRYP handle. <code>__FLAG__</code>: specifies the flag to check. This parameter can be one of the following values: <code>CRYP_FLAG_CCF</code> : Computation Complete Flag <code>CRYP_FLAG_RDERR</code> : Read Error Flag <code>CRYP_FLAG_WRERR</code> : Write Error Flag Return value: - The: new state of <code>__FLAG__</code> (TRUE or

__HAL_CRYPT_CLEAR_FLAG

FALSE).

Description:

- Clear the CRYPT pending flag.

Parameters:

- **__HANDLE__**: specifies the CRYPT handle.
- **__FLAG__**: specifies the flag to clear. This parameter can be one of the following values:
 - **CRYPT_CLEARFLAG_CCF** : Computation Complete Clear Flag
 - **CRYPT_CLEARFLAG_RDERR** : Read Error Clear
 - **CRYPT_CLEARFLAG_WRERR** : Write Error Clear

Return value:

- None

__HAL_CRYPT_ENABLE_IT**Description:**

- Enable the CRYPT interrupt.

Parameters:

- **__HANDLE__**: specifies the CRYPT handle.
- **__INTERRUPT__**: CRYPT Interrupt.

Return value:

- None

__HAL_CRYPT_DISABLE_IT**Description:**

- Disable the CRYPT interrupt.

Parameters:

- **__HANDLE__**: specifies the CRYPT handle.
- **__INTERRUPT__**: CRYPT interrupt.

Return value:

- None

__HAL_CRYPT_GET_IT_SOURCE**Description:**

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

Parameters:

- **__HANDLE__**: specifies the CRYPT handle.
- **__INTERRUPT__**: CRYPT interrupt source to check This parameter can be one of the following values:
 - **CRYPT_IT_CC** : Computation

`__HAL_CRYPT_CLEAR_IT`

Complete interrupt

- CRYPT_IT_ERR : Error interrupt
(used for RDERR and WRERR)

Return value:

- State: of interruption (SET or RESET)

Description:

- Clear the CRYPT pending IT.

Parameters:

- `__HANDLE__`: specifies the CRYPT handle.
- `__IT__`: specifies the IT to clear. This parameter can be one of the following values:
 - CRYPT_CLEARFLAG_CCF :
Computation Complete Clear Flag
 - CRYPT_CLEARFLAG_RDERR :
Read Error Clear
 - CRYPT_CLEARFLAG_WRERR :
Write Error Clear

Return value:

- None

12 HAL CRYPT Extension Driver

12.1 CRYPTEx Firmware driver API description

12.1.1 Extended features functions

This section provides callback functions:

- Computation completed.

This section contains the following APIs:

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

12.1.2 HAL_CRYPTEx_ComputationCpltCallback

Function Name	void HAL_CRYPTEx_ComputationCpltCallback (CRYPT_HandleTypeDef * hcryp)
Function Description	Computation completed callbacks.
Parameters	• hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	• None:

13 HAL DAC Generic Driver

13.1 DAC Firmware driver registers structures

13.1.1 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

13.1.2 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](#)

13.2 DAC Firmware driver API description

13.2.1 DAC Peripheral features

DAC Channels

STM32F0 devices integrates no, one or two 12-bit Digital Analog Converters. STM32L05x & STM32L06x devices have one converter (channel1) STM32L07x & STM32L08x 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 (STM32L07x/STM32L08x only)
3. Channel1 & channel2 can be used independently or simultaneously in dual mode (STM32L07x/STM32L08x only)

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_Trigger_None and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_Pin9) using DAC_Trigger_Ext_IT9. The used pin (GPIOx_Pin9) must be configured in input mode.
2. Timers TRGO: STM32L05x/STM32L06x : TIM2, TIM6 and TIM21
STM32L07x/STM32L08x : TIM2, TIM3, TIM6, TIM7 and TIM21
(DAC_Trigger_T2_TRGO, DAC_Trigger_T6_TRGO...)
3. Software using DAC_Trigger_Software

DAC Buffer mode feature

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



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

DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave using `HAL_DACEx_NoiseWaveGenerate()`
2. Triangle wave using `HAL_DACEx_TriangleWaveGenerate()`

DAC data format

The DAC data format can be:

1. 8-bit right alignment using `DAC_ALIGN_8B_R`
2. 12-bit left alignment using `DAC_ALIGN_12B_L`
3. 12-bit right alignment using `DAC_ALIGN_12B_R`

DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation: $DAC\_OUTx = VREF+ * DOR / 4095$ with DOR is the Data Output Register VEF+ is the input voltage reference (refer to the device datasheet) e.g. To set DAC_OUT1 to 0.7V, use Assuming that $VREF+ = 3.3V$, $DAC\_OUT1 = (3.3 * 868) / 4095 = 0.7V$

DMA requests

A DMA1 request can be generated when an external trigger (but not a software trigger) occurs if DMA1 requests are enabled using HAL_DAC_Start_DMA()

DMA1 requests are mapped as following:

1. DAC channel1 : mapped on DMA1 Request9 channel2 which must be already configured
2. DAC channel2 : mapped on DMA1 Request15 channel4 which must be already configured (STM32L07x/STM32L08x only) For Dual mode (STM32L07x/STM32L08x only) and specific signal Triangle and noise generation please refer to Extension Features Driver description

13.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL_DAC_Init()
- Configure DAC_OUT1: PA4 in analog mode.
- Configure DAC_OUT2: PA5 in analog mode (STM32L07x/STM32L08x only).
- Configure the DAC channel using HAL_DAC_ConfigChannel() function.
- Enable the DAC channel using HAL_DAC_Start() or HAL_DAC_Start_DMA functions

Polling mode IO operation

- Start the DAC peripheral using HAL_DAC_Start()
- To read the DAC last data output value value, use the HAL_DAC_GetValue() function.
- Stop the DAC peripheral using HAL_DAC_Stop()

DMA mode IO operation

- Start the DAC peripheral using HAL_DAC_Start_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion
- At the middle of data transfer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DAC_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1 or HAL_DAC_ConvHalfCpltCallbackCh2
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1() or HAL_DAC_ConvCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1 or HAL_DAC_ConvCpltCallbackCh2
- In case of transfer Error, HAL_DAC_ErrorCallbackCh1() 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_DAC_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1 or HAL_DAC_DMAUnderrunCallbackCh2 add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1
- Stop the DAC peripheral using HAL_DAC_Stop_DMA()

DAC HAL driver macros list

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

- __HAL_DAC_ENABLE : Enable the DAC peripheral
- __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

13.2.3 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_DeInit\(\)*](#)
- [*HAL_DAC_MspInit\(\)*](#)
- [*HAL_DAC_MspDeInit\(\)*](#)

13.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion (STM32L07xx/STM32L08xx only)

This section contains the following APIs:

- [*HAL_DAC_Start\(\)*](#)
- [*HAL_DAC_Stop\(\)*](#)
- [*HAL_DAC_Start_DMA\(\)*](#)
- [*HAL_DAC_Stop_DMA\(\)*](#)
- [*HAL_DAC_GetValue\(\)*](#)
- [*HAL_DAC_IRQHandler\(\)*](#)
- [*HAL_DAC_ConvCpltCallbackCh1\(\)*](#)
- [*HAL_DAC_ConvHalfCpltCallbackCh1\(\)*](#)
- [*HAL_DAC_ErrorCallbackCh1\(\)*](#)

- [HAL_DAC_DMAUnderrunCallbackCh1\(\)](#)
- [HAL_DAC_SetValue\(\)](#)

13.2.5 Peripheral Control functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [HAL_DAC_ConfigChannel\(\)](#)

13.2.6 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\(\)](#)
- [HAL_DAC_SetValue\(\)](#)

13.2.7 HAL_DAC_Init

Function Name	HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)
Function Description	Initializes the DAC peripheral according to the specified parameters in the DAC_InitStruct.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• HAL: status

13.2.8 HAL_DAC_DeInit

Function Name	HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)
Function Description	Deinitializes the DAC peripheral registers to their default reset values.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• HAL: status

13.2.9 HAL_DAC_Msplnit

Function Name	void HAL_DAC_Msplnit (DAC_HandleTypeDef * hdac)
Function Description	Initializes the DAC MSP.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

13.2.10 HAL_DAC_MspDeInit

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

Function Description DeInitializes the DAC MSP.

Parameters

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

Return values

- **None:**

13.2.11 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

- **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

13.2.12 HAL_DAC_Stop

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

Function Description Disables DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)

Return values

- **HAL:** status

13.2.13 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 using DMA.

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 (STM32L07x/STM32L08x only)
- **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

13.2.14 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 (STM32L07x/STM32L08x only)
- Return values
- **HAL:** status

13.2.15 HAL_DAC_GetValue

- Function Name **uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)**
- Function Description Returns the last data output value of the selected DAC channel.
- Parameters
- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
 - **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)
- Return values
- **The:** selected DAC channel data output value.

13.2.16 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:**

13.2.17 HAL_DAC_ConvCpltCallbackCh1

- Function Name **void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)**

Function Description	Conversion complete callback in non blocking mode for Channel1.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None:

13.2.18 HAL_DAC_ConvHalfCpltCallbackCh1

Function Name	void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)
Function Description	Conversion half DMA transfer callback in non blocking mode for Channel1.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None:

13.2.19 HAL_DAC_ErrorCallbackCh1

Function Name	void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)
Function Description	Error DAC callback for Channel1.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None:

13.2.20 HAL_DAC_DMAUnderrunCallbackCh1

Function Name	void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)
Function Description	DMA underrun DAC callback for channel1.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None:

13.2.21 HAL_DAC_SetValue

Function Name	HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)
Function Description	Set the specified data holding register value for DAC channel.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only) • 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.
- **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.

Return values

- **HAL:** status
- **HAL:** status

13.2.22 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**
- **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 (STM32L07x/STM32L08x only)

Return values

- **HAL:** status

13.2.23 HAL_DAC_GetState

Function Name HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)

Function Description return the DAC state

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

Return values

- **HAL:** state

13.2.24 HAL_DAC_GetError

Function Name uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)

Function Description Return the DAC error code.

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

Return values

- **DAC:** Error Code

13.2.25 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 (STM32L07x/STM32L08x only) • 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.
Return values	• HAL: status • HAL: status

13.3 DAC Firmware driver defines**13.3.1 DAC****DAC Channel selection**

DAC_CHANNEL_1

DAC_CHANNEL_2

IS_DAC_CHANNEL

DAC data

IS_DAC_DATA

DAC data alignment

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

IS_DAC_ALIGN
DAC Error Code

HAL_DAC_ERROR_NONE	No error
HAL_DAC_ERROR_DMAUNDERRUNCH1	DAC channel1 DAM underrun error
HAL_DAC_ERROR_DMAUNDERRUNCH2	DAC channel2 DAM 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: - Disable the DAC interrupt. Parameters: __HANDLE__: specifies the DAC handle __INTERRUPT__: specifies the DAC interrupt. Return value: - None
__HAL_DAC_DISABLE_IT	Description: - Disable the DAC interrupt. Parameters: __HANDLE__: specifies the DAC handle __INTERRUPT__: specifies the DAC interrupt. Return value: - None

__HAL_DAC_GET_IT_SOURCE**Description:**

- 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 (STM32L072xx STM32L073xx STM32L082xx STM32L083xx only)

Return value:

- State: of interruption (SET or RESET)

__HAL_DAC_GET_FLAG**Description:**

- Get the selected DAC's flag status.

Parameters:

- **__HANDLE__**: specifies the DAC handle.
- **__FLAG__**: specifies the FLAG.

Return value:

- None

__HAL_DAC_CLEAR_FLAG**Description:**

- Clear the DAC's flag.

Parameters:

- **__HANDLE__**: specifies the DAC handle.
- **__FLAG__**: specifies the 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

IS_DAC_OUTPUT_BUFFER_STATE

DAC trigger selection

DAC_TRIGGER_NONE	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
DAC_TRIGGER_T6_TRGO	TIM6 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T21_TRGO	TIM21 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T2_TRGO	TIM2 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_EXT_IT9	EXTI Line9 event selected as external conversion trigger for DAC channel
DAC_TRIGGER_SOFTWARE	Conversion started by software trigger for DAC channel
DAC_TRIGGER_T3_TRGO	TIM3 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T3_CH3	TIM3 CH3 selected as external conversion trigger for DAC channel
DAC_TRIGGER_T7_TRGO	TIM7 TRGO selected as external conversion trigger for DAC channel
IS_DAC_TRIGGER	

14 HAL DAC Extension Driver

14.1 DACEx Firmware driver API description

14.1.1 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.

14.1.2 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.

14.1.3 HAL_DACEx_TriangleWaveGenerate

Function Name	HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only) • 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

- **HAL:** status

14.1.4 HAL_DACEx_NoiseWaveGenerate

Function Name

HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate
(DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t
Amplitude)

Function Description

Enables or disables the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)
- **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

14.1.5 HAL_DACEx_DualSetValue

Function Name

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

Function Description

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

Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Alignment: Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected • Data1: Data for DAC Channel2 to be loaded in the selected data holding register. • Data2: Data for DAC Channel1 to be loaded in the selected data holding register.
Return values	• HAL: status
Notes	• In dual mode, a unique register access is required to write in both DAC channels at the same time.

14.1.6 HAL_DACEx_ConvCpltCallbackCh2

Function Name	void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)
Function Description	Conversion complete callback in non blocking mode for Channel2.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None:

14.1.7 HAL_DACEx_ConvHalfCpltCallbackCh2

Function Name	void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)
Function Description	Conversion half DMA transfer callback in non blocking mode for Channel2.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None:

14.1.8 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:

14.1.9 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:

14.1.10 HAL_DAC_Start

Function Name	HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef *hdac, uint32_t Channel)
Function Description	Enables DAC and starts conversion of channel.
Parameters	• hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	• HAL: status

14.1.11 HAL_DAC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef *hdac, uint32_t Channel, uint32_t *pData, uint32_t Length, uint32_t Alignment)
Function Description	Enables DAC and starts conversion of channel using DMA.
Parameters	• 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 (STM32L07x/STM32L08x only) • 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

14.1.12 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 (STM32L07x/STM32L08x only)

Return values

- **HAL:** status

14.1.13 HAL_DAC_GetValue

Function Name **uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)**

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

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected (STM32L07x/STM32L08x only)

Return values

- **The:** selected DAC channel data output value.

14.1.14 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:**

14.1.15 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 (STM32L07x/STM32L08x only)
- **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.
- **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.

Return values

- **HAL:** status
- **HAL:** status

14.2 DACEx Firmware driver defines

14.2.1 DACEx

DACEx lfsrunmask 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
IS_DAC_LFSR_UNMASK_TRIANGLE_AMPLITUDE	

15 HAL DMA Generic Driver

15.1 DMA Firmware driver registers structures

15.1.1 DMA_InitTypeDef

Data Fields

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

Field Documentation

- ***uint32_t DMA_InitTypeDef::Request***
Specifies the request selected for the specified channel. This parameter can be a value of [DMA_request](#)
- ***uint32_t DMA_InitTypeDef::Direction***
Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of [DMA_Data_transfer_direction](#)
- ***uint32_t DMA_InitTypeDef::PeriphInc***
Specifies whether the Peripheral address register should be incremented or not. When Memory to Memory transfer is used, this is the Source Increment mode 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. When Memory to Memory transfer is used, this is the Destination Increment mode This parameter can be a value of [DMA_Memory_incremented_mode](#)
- ***uint32_t DMA_InitTypeDef::PeriphDataAlignment***
Specifies the Peripheral data width. When Memory to Memory transfer is used, this is the Source Alignment format This parameter can be a value of [DMA_Peripheral_data_size](#)
- ***uint32_t DMA_InitTypeDef::MemDataAlignment***
Specifies the Memory data width. When Memory to Memory transfer is used, this is the Destination Alignment format This parameter can be a value of [DMA_Memory_data_size](#)
- ***uint32_t DMA_InitTypeDef::Mode***
Specifies the operation mode of the DMAy Channelx (Normal or Circular). 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](#)

15.1.2 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

15.2 DMA Firmware driver API description

15.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DMA
- De-Initialize the DMA

This section contains the following APIs:

- [*HAL_DMA_Init\(\)*](#)
- [*HAL_DMA_DeInit\(\)*](#)

15.2.2 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt
- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request

This section contains the following APIs:

- [HAL_DMA_Start\(\)](#)
- [HAL_DMA_Start_IT\(\)](#)
- [HAL_DMA_Abort\(\)](#)
- [HAL_DMA_PollForTransfer\(\)](#)
- [HAL_DMA_IRQHandler\(\)](#)

15.2.3 Peripheral State 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\(\)](#)

15.2.4 HAL_DMA_Init

Function Name	HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)
Function Description	Initializes the DMA according to the specified parameters in the DMA_InitTypeDef and create the associated handle.
Parameters	• hdma: Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• HAL: status

15.2.5 HAL_DMA_DeInit

Function Name	HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef * hdma)
Function Description	DeInitializes the DMA peripheral.
Parameters	• hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• HAL: status

15.2.6 HAL_DMA_Start

Function Name	HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef *

hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)

Function Description Starts the DMA Transfer.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA 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

- **HAL:** status

15.2.7 HAL_DMA_Start_IT

Function Name **HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)**

Function Description Start the DMA Transfer with interrupt enabled.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA 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

- **HAL:** status

15.2.8 HAL_DMA_Abort

Function Name **HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)**

Function Description Aborts the DMA Transfer.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status

15.2.9 HAL_DMA_PollForTransfer

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

Function Description Polling for transfer complete.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CompleteLevel:** Specifies the DMA level complete.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

15.2.10 HAL_DMA_IRQHandler

Function Name void HAL_DMA_IRQHandler (DMA_HandleTypeDef * hdma)

Function Description Handles DMA interrupt request.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **None:**

15.2.11 HAL_DMA_GetState

Function Name HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef * hdma)

Function Description Returns the DMA state.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** state

15.2.12 HAL_DMA_GetError

Function Name uint32_t HAL_DMA_GetError (DMA_HandleTypeDef * hdma)

Function Description Return the DMA error code.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **DMA:** Error Code

15.3 DMA Firmware driver defines

15.3.1 DMA

DMA Data Buffer Size Check

IS_DMA_BUFFER_SIZE

DMA Data Transfer directions

DMA_PERIPH_TO_MEMORY Peripheral to memory direction

DMA_MEMORY_TO_PERIPH Memory to peripheral direction

DMA_MEMORY_TO_MEMORY Memory to memory direction

IS_DMA_DIRECTION

DMA Error Codes

HAL_DMA_ERROR_NONE No error

HAL_DMA_ERROR_TE Transfer error

HAL_DMA_ERROR_TIMEOUT Timeout error

IS_DMA_ALL_INSTANCE

IS_DMA_ALL_CONTROLLER

DMA Exported Macros

__HAL_DMA_RESET_HANDLE_STATE

Description:

- Reset DMA handle state.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_DMA_ENABLE

Description:

- Enable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None.

__HAL_DMA_DISABLE

Description:

- Disable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None.

__HAL_DMA_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_GI_FLAG_INDEX`

Description:

- Returns the current DMA Channel Global interrupt flag.

Parameters:

- `__HANDLE__`: DMA handle

Return value:

- The: specified transfer error flag index.

`__HAL_DMA_GET_FLAG`

Description:

- Get the DMA Channel pending flags.

Parameters:

- `__HANDLE__`: DMA handle
- `__FLAG__`: Get the specified flag. This parameter can be any combination of the following values:
 - `DMA_FLAG_TCIFx`: Transfer complete flag
 - `DMA_FLAG_HTIFx`: Half transfer complete flag
 - `DMA_FLAG_TEIFx`: Transfer error flag
 - `DMA_ISR_GIFx`: Global interrupt flag
 Where x can be 0_4, 1_5, 2_6 or 3_7 to select the DMA Channel flag.

Return value:

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

`__HAL_DMA_CLEAR_FLAG`

Description:

- Clears the DMA Channel pending flags.

Parameters:

- `__HANDLE__`: DMA handle
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
 - `DMA_FLAG_TCIFx`: Transfer complete flag
 - `DMA_FLAG_HTIFx`: Half transfer

- complete flag
 - DMA_FLAG_TEIFx: Transfer error flag
 - DMA_ISR_GIFx: Global interrupt flag
- Where x can be 0_4, 1_5, 2_6 or 3_7 to select the DMA Channel flag.

Return value:

- None

Description:

- Enables the specified DMA Channel interrupts.

Parameters:

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

Return value:

- None

Description:

- Disables the specified DMA Channel interrupts.

Parameters:

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

Return value:

- None

Description:

- Checks whether the specified DMA Channel interrupt is enabled or not.

__HAL_DMA_ENABLE_IT

__HAL_DMA_DISABLE_IT

__HAL_DMA_GET_IT_SOURCE

Parameters:

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

Return value:

- The: state of `DMA_IT` (SET or RESET).

DMA Flag Definitions`DMA_FLAG_GL1``DMA_FLAG_TC1``DMA_FLAG_HT1``DMA_FLAG_TE1``DMA_FLAG_GL2``DMA_FLAG_TC2``DMA_FLAG_HT2``DMA_FLAG_TE2``DMA_FLAG_GL3``DMA_FLAG_TC3``DMA_FLAG_HT3``DMA_FLAG_TE3``DMA_FLAG_GL4``DMA_FLAG_TC4``DMA_FLAG_HT4``DMA_FLAG_TE4``DMA_FLAG_GL5``DMA_FLAG_TC5``DMA_FLAG_HT5``DMA_FLAG_TE5``DMA_FLAG_GL6``DMA_FLAG_TC6``DMA_FLAG_HT6``DMA_FLAG_TE6``DMA_FLAG_GL7`

DMA_FLAG_TC7

DMA_FLAG_HT7

DMA_FLAG_TE7

DMA 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_TRIGGER Index of the DMA handle used for Trigger DMA requests

DMA Interrupt Definitions

DMA_IT_TC

DMA_IT_HT

DMA_IT_TE

DMA Memory Data Size Alignment

DMA_MDATAALIGN_BYTE Memory data alignment : Byte

DMA_MDATAALIGN_HALFWORD Memory data alignment : HalfWord

DMA_MDATAALIGN_WORD Memory data alignment : Word

IS_DMA_MEMORY_DATA_SIZE

DMA Memory Incremented Mode

DMA_MINC_ENABLE Memory increment mode Enable

DMA_MINC_DISABLE Memory increment mode Disable

IS_DMA_MEMORY_INC_STATE

DMA Mode

DMA_NORMAL Normal Mode

DMA_CIRCULAR Circular Mode

IS_DMA_MODE

DMA Peripheral Data Size Alignment

DMA_PDATAALIGN_BYTE Peripheral data alignment : Byte

DMA_PDATAALIGN_HALFWORD Peripheral data alignment : HalfWord

DMA_PDATAALIGN_WORD Peripheral data alignment : Word

IS_DMA_PERIPHERAL_DATA_SIZE

DMA Peripheral Incremented Mode

DMA_PINC_ENABLE Peripheral increment mode Enable

DMA_PINC_DISABLE Peripheral increment mode Disable

IS_DMA_PERIPHERAL_INC_STATE

DMA Priority Level

DMA_PRIORITY_LOW Priority level : Low

DMA_PRIORITY_MEDIUM Priority level : Medium

DMA_PRIORITY_HIGH Priority level : High

DMA_PRIORITY_VERY_HIGH Priority level : Very_High

IS_DMA_PRIORITY

DMA request definitions

DMA_REQUEST_0

DMA_REQUEST_1

DMA_REQUEST_2

DMA_REQUEST_3

DMA_REQUEST_4

DMA_REQUEST_5

DMA_REQUEST_6

DMA_REQUEST_7

DMA_REQUEST_8

DMA_REQUEST_9

DMA_REQUEST_10

DMA_REQUEST_11

DMA_REQUEST_12

DMA_REQUEST_13

DMA_REQUEST_14

DMA_REQUEST_15

IS_DMA_ALL_REQUEST

16 HAL FIREWALL Generic Driver

16.1 FIREWALL Firmware driver registers structures

16.1.1 FIREWALL_InitTypeDef

Data Fields

- *uint32_t CodeSegmentStartAddress*
- *uint32_t CodeSegmentLength*
- *uint32_t NonVDataSegmentStartAddress*
- *uint32_t NonVDataSegmentLength*
- *uint32_t VDataSegmentStartAddress*
- *uint32_t VDataSegmentLength*
- *uint32_t VolatileDataExecution*
- *uint32_t VolatileDataShared*

Field Documentation

- *uint32_t FIREWALL_InitTypeDef::CodeSegmentStartAddress*
Protected code segment start address. This value is 24-bit long, the 8 LSB bits are reserved and forced to 0 in order to allow a 256-byte granularity.
- *uint32_t FIREWALL_InitTypeDef::CodeSegmentLength*
Protected code segment length in bytes. This value is 22-bit long, the 8 LSB bits are reserved and forced to 0 for the length to be a multiple of 256 bytes.
- *uint32_t FIREWALL_InitTypeDef::NonVDataSegmentStartAddress*
Protected non-volatile data segment start address. This value is 24-bit long, the 8 LSB bits are reserved and forced to 0 in order to allow a 256-byte granularity.
- *uint32_t FIREWALL_InitTypeDef::NonVDataSegmentLength*
Protected non-volatile data segment length in bytes. This value is 22-bit long, the 8 LSB bits are reserved and forced to 0 for the length to be a multiple of 256 bytes.
- *uint32_t FIREWALL_InitTypeDef::VDataSegmentStartAddress*
Protected volatile data segment start address. This value is 17-bit long, the 6 LSB bits are reserved and forced to 0 in order to allow a 64-byte granularity.
- *uint32_t FIREWALL_InitTypeDef::VDataSegmentLength*
Protected volatile data segment length in bytes. This value is 17-bit long, the 6 LSB bits are reserved and forced to 0 for the length to be a multiple of 64 bytes.
- *uint32_t FIREWALL_InitTypeDef::VolatileDataExecution*
Set VDE bit specifying whether or not the volatile data segment can be executed. When VDS = 1 (set by parameter *VolatileDataShared*), VDE bit has no meaning. This parameter can be a value of [*FIREWALL_VolatileData_Executable*](#)
- *uint32_t FIREWALL_InitTypeDef::VolatileDataShared*
Set VDS bit in specifying whether or not the volatile data segment can be shared with a non-protected application code. This parameter can be a value of [*FIREWALL_VolatileData_Shared*](#)

16.2 FIREWALL Firmware driver API description

16.2.1 How to use this driver

The FIREWALL HAL driver can be used as follows:

1. Declare a `FIREWALL_InitTypeDef` initialization structure.
2. Resort to `HAL_FIREWALL_Config()` API to initialize the Firewall
3. Enable the FIREWALL in calling `HAL_FIREWALL_EnableFirewall()` API
4. To ensure that any code executed outside the protected segment closes the FIREWALL, the user must set the flag `FIREWALL_PRE_ARM_SET` in calling `__HAL_FIREWALL_PREARM_ENABLE()` macro if called within a protected code segment or `HAL_FIREWALL_EnablePreArmFlag()` API if called outside of protected code segment after `HAL_FIREWALL_Config()` call.

16.2.2 Initialization and Configuration functions

This subsection provides the functions allowing to initialize the Firewall. Initialization is done by `HAL_FIREWALL_Config()`:

- Enable the Firewall clock thru `__HAL_RCC_FIREWALL_CLK_ENABLE()` macro.
- Set the protected code segment address start and length.
- Set the protected non-volatile and/or volatile data segments address starts and lengths if applicable.
- Set the volatile data segment execution and sharing status.
- Length must be set to 0 for an unprotected segment.

This section contains the following APIs:

- [`HAL\_FIREWALL\_Config\(\)`](#)
- [`HAL\_FIREWALL\_GetConfig\(\)`](#)
- [`HAL\_FIREWALL\_EnableFirewall\(\)`](#)
- [`HAL\_FIREWALL\_EnablePreArmFlag\(\)`](#)
- [`HAL\_FIREWALL\_DisablePreArmFlag\(\)`](#)

16.2.3 HAL_FIREWALL_Config

Function Name	<code>HAL_StatusTypeDef HAL_FIREWALL_Config (FIREWALL_InitTypeDef * fw_init)</code>
Function Description	Initialize the Firewall according to the <code>FIREWALL_InitTypeDef</code> structure parameters.
Parameters	• fw_init: Firewall initialization structure
Return values	• HAL: status
Notes	• The API returns <code>HAL_ERROR</code> if the Firewall is already enabled.

16.2.4 HAL_FIREWALL_GetConfig

Function Name	<code>void HAL_FIREWALL_GetConfig (FIREWALL_InitTypeDef * fw_config)</code>
Function Description	Retrieve the Firewall configuration.
Parameters	• fw_config: Firewall configuration, type is same as initialization structure

Return values	• None:
Notes	• This API can't be executed inside a code area protected by the Firewall when the Firewall is enabled • If NVDSL register is different from 0, that is, if the non volatile data segment is defined, this API can't be executed when the Firewall is enabled. • User should resort to <code>__HAL_FIREWALL_GET_PREARM()</code> macro to retrieve FPA bit status

16.2.5 HAL_FIREWALL_EnableFirewall

Function Name	void HAL_FIREWALL_EnableFirewall (void)
Function Description	Enable FIREWALL.
Return values	• None:
Notes	• Firewall is enabled in clearing FWDIS bit of SYSCFG CFGR1 register. Once enabled, the Firewall cannot be disabled by software. Only a system reset can set again FWDIS bit.

16.2.6 HAL_FIREWALL_EnablePreArmFlag

Function Name	void HAL_FIREWALL_EnablePreArmFlag (void)
Function Description	Enable FIREWALL pre arm.
Return values	• None:
Notes	• When FPA bit is set, any code executed outside the protected segment will close the Firewall. • This API provides the same service as <code>__HAL_FIREWALL_PREARM_ENABLE()</code> macro but can't be executed inside a code area protected by the Firewall. • When the Firewall is disabled, user can resort to <code>HAL_FIREWALL_EnablePreArmFlag()</code> API any time. • When the Firewall is enabled and NVDSL register is equal to 0 (that is, when the non volatile data segment is not defined), ** this API can be executed when the Firewall is closed ** when the Firewall is opened, user should resort to <code>__HAL_FIREWALL_PREARM_ENABLE()</code> macro instead • When the Firewall is enabled and NVDSL register is different from 0 (that is, when the non volatile data segment is defined) ** FW_CR register can be accessed only when the Firewall is opened: user should resort to <code>__HAL_FIREWALL_PREARM_ENABLE()</code> macro instead.

16.2.7 HAL_FIREWALL_DisablePreArmFlag

Function Name	void HAL_FIREWALL_DisablePreArmFlag (void)
Function Description	Disable FIREWALL pre arm.
Return values	• None:
Notes	• When FPA bit is reset, any code executed outside the protected segment when the Firewall is opened will generate a system reset.

- This API provides the same service as `__HAL_FIREWALL_PREARM_DISABLE()` macro but can't be executed inside a code area protected by the Firewall.
- When the Firewall is disabled, user can resort to `HAL_FIREWALL_EnablePreArmFlag()` API any time.
- When the Firewall is enabled and NVDSL register is equal to 0 (that is, when the non volatile data segment is not defined), ** this API can be executed when the Firewall is closed ** when the Firewall is opened, user should resort to `__HAL_FIREWALL_PREARM_DISABLE()` macro instead
- When the Firewall is enabled and NVDSL register is different from 0 (that is, when the non volatile data segment is defined) ** FW_CR register can be accessed only when the Firewall is opened: user should resort to `__HAL_FIREWALL_PREARM_DISABLE()` macro instead.

16.3 FIREWALL Firmware driver defines

16.3.1 FIREWALL

FIREWALL Exported Macros

`__HAL_FIREWALL_IS_ENABLED`

Description:

- Check whether the FIREWALL is enabled or not.

Return value:

- FIREWALL: enabling status (TRUE or FALSE).

`__HAL_FIREWALL_PREARM_ENABLE`

Notes:

- When FPA bit is set, any code executed outside the protected segment closes the Firewall, otherwise it generates a system reset. This macro provides the same service as `HAL_FIREWALL_EnablePreArmFlag()` API but can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

`__HAL_FIREWALL_PREARM_DISABLE`

Notes:

- When FPA bit is set, any code executed outside the protected segment closes the Firewall, otherwise, it generates a system reset. This macro provides the same service as `HAL_FIREWALL_DisablePreArmFlag()` API but can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

`__HAL_FIREWALL_VOLATILEDATA_SHARED_ENABLE`

Notes:

- When VDS bit is set, the volatile data segment is shared with non-protected application code. It can be accessed whatever the Firewall state (opened or closed). This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

`__HAL_FIREWALL_VOLATILEDATA_SHARED_DISABLE`

Notes:

- When VDS bit is reset, the volatile data segment is not shared and cannot be hit by a non protected executable code when the Firewall is closed. If it is accessed in such a condition, a system reset is generated by the Firewall. This macro can be executed inside a code area protected by the Firewall.

This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

`__HAL_FIREWALL_VOLATILEDATA_EXECUTION_ENABLE`

Notes:

- VDE bit is ignored when VDS is set. IF VDS = 1, the Volatile data segment can be executed whatever the VDE bit value. When VDE bit is set (with VDS = 0), the volatile data segment is executable. When the Firewall call is closed, a "call gate" entry procedure is required to open first the Firewall. This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

`__HAL_FIREWALL_VOLATILEDATA_EXECUTION_DISABLE`

Notes:

- VDE bit is ignored when VDS is set. IF VDS = 1, the Volatile data segment can be executed whatever the VDE bit value. When VDE bit is reset (with VDS = 0), the volatile data segment cannot be executed. This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0,

that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_GET_VOLATILEDATA_SHARED

Description:

- Check whether or not the volatile data segment is shared.

Return value:

- VDS: bit setting status (TRUE or FALSE).

Notes:

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_GET_VOLATILEDATA_EXECUTION

Description:

- Check whether or not the volatile data segment is declared executable.

Return value:

- VDE: bit setting status (TRUE or FALSE).

Notes:

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

__HAL_FIREWALL_GET_PREARM

Description:

- Check whether or not the

Firewall pre arm bit is set.

Return value:

- FPA: bit setting status (TRUE or FALSE).

Notes:

- This macro can be executed inside a code area protected by the Firewall. This macro can be executed whatever the Firewall state (opened or closed) when NVDSL register is equal to 0. Otherwise (when NVDSL register is different from 0, that is, when the non volatile data segment is defined), the macro can be executed only when the Firewall is opened.

FIREWALL pre arm status

FIREWALL_PRE_ARM_RESET

FIREWALL_PRE_ARM_SET

FIREWALL volatile data segment execution status

FIREWALL_VOLATILEDATA_NOT_EXECUTABLE

FIREWALL_VOLATILEDATA_EXECUTABLE

FIREWALL volatile data segment share status

FIREWALL_VOLATILEDATA_NOT_SHARED

FIREWALL_VOLATILEDATA_SHARED

17 HAL FLASH Generic Driver

17.1 FLASH 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: Page Erase only. This parameter can be a value of [FLASHEx_Type_Erase](#)
- *uint32_t FLASH_EraseInitTypeDef::PageAddress*
PageAddress : Initial FLASH address to be erased This parameter must be a value belonging to FLASH Programm address (depending on the devices)
- *uint32_t FLASH_EraseInitTypeDef::NbPages*
NbPages: Number of pages to be erased. This parameter must be a value between 1 and (max number of pages - value of Initial page)

17.1.2 FLASH_ProcessTypeDef

Data Fields

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

Field Documentation

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

17.2 FLASH Firmware driver API description

17.2.1 FLASH peripheral features

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

The Flash memory interface accelerates code execution with a system of instruction prefetch.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Prefetch on I-Code
- Option Bytes programming

17.2.2 How to use this driver

This driver provides functions to configure and program the Flash memory of all STM32L0xx devices.

1. FLASH Memory 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.
 - Program functions: Fast Word and Half Page(should be executed from internal SRAM).
2. DATA EEPROM Programming functions: this group includes all needed functions to erase and program the DATA EEPROM memory:
 - Lock and Unlock the DATA EEPROM interface.
 - Erase function: Erase Byte, erase HalfWord, erase Word, erase Double Word (should be executed from internal SRAM).
 - Program functions: Fast Program Byte, Fast Program Half-Word, FastProgramWord, Program Byte, Program Half-Word, Program Word and Program Double-Word (should be executed from internal SRAM).
3. FLASH Option Bytes Programming functions: this group includes all needed functions to:
 - Lock and Unlock the Flash Option bytes.
 - Set/Reset the write protection.
 - Set the Read protection Level.
 - Set the BOR level.
 - Program the user option Bytes.
 - Launch the Option Bytes loader.
 - Get the Write protection.
 - Get the read protection status.
 - Get the BOR level.
 - Get the user option bytes.
4. Interrupts and flags management functions :
 - Handle FLASH interrupts by calling HAL_FLASH_IRQHandler()
 - Wait for last FLASH operation according to its status
 - Get error flag status by calling HAL_GetErrorCode()
5. FLASH Interface configuration functions: this group includes the management of following features:

- Enable/Disable the RUN PowerDown mode.
 - Enable/Disable the SLEEP PowerDown mode.
6. FLASH Peripheral State methods: this group includes the management of following features:
- Wait for the FLASH operation
 - Get the specific FLASH error flag

In addition to these function, this driver includes a set of macros allowing to handle the following operations:

- Set/Get the latency
- Enable/Disable the prefetch buffer
- Enable/Disable the preread buffer
- Enable/Disable the Flash power-down
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

=====

=====

17.2.3 Programming operation functions

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

The FLASH Memory Programming functions, includes the following functions:

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

Any operation of erase or program should follow these steps:

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

=====

=====

17.2.4 Option Bytes Programming functions

The FLASH_Option Bytes Programming_functions, includes the following functions:

- HAL_FLASH_OB_Unlock(void);
- HAL_FLASH_OB_Lock(void);
- HAL_FLASH_OB_Launch(void);
- HAL_FLASHEx_OBProgram(FLASH_OBProgramInitTypeDef *pOBInit);
- HAL_FLASHEx_OBGetConfig(FLASH_OBProgramInitTypeDef *pOBInit);

Any operation of erase or program should follow these steps:

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

4. Call the HAL_FLASH_OB_Lock() to disable the Flash option control register access (recommended to protect the option Bytes against possible unwanted operations).

Proprietary code Read Out Protection (PcROP):

1. The PcROP sector is selected by using the same option bytes as the Write protection. As a result, these 2 options are exclusive each other.
2. To activate PCROP mode for Flash sectors(s), you need to follow the sequence below:
 - Use this function HAL_FLASHEx_AdvOBProgram with PCROPState = OB_PCROP_STATE_ENABLE. *

17.2.5 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\(\)](#)

17.2.6 Peripheral Errors functions

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

This section contains the following APIs:

- [HAL_FLASH_GetError\(\)](#)

17.2.7 HAL_FLASH_Program

Function Name	HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint32_t Data)
Function Description	Program word at a specified address.
Parameters	• TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program • Address: specifies the address to be programmed. • Data: specifies the data to be programmed
Return values	• HAL_StatusTypeDef: HAL Status
Notes	• To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation).

17.2.8 HAL_FLASH_Program_IT

Function Name	HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint32_t Data)
Function Description	Program word at a specified address with interrupt enabled.
Parameters	• TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type

	Program
	• Address: specifies the address to be programmed. • Data: specifies the data to be programmed
Return values	• HAL_StatusTypeDef: HAL Status
17.2.9	HAL_FLASH_IRQHandler
Function Name	void HAL_FLASH_IRQHandler (void)
Function Description	This function handles FLASH interrupt request.
Return values	• None:
17.2.10	HAL_FLASH_EndOfOperationCallback
Function Name	void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)
Function Description	FLASH end of operation interrupt callback.
Parameters	• ReturnValue: The value saved in this parameter depends on the ongoing procedurePages Erase: Sector which has been erased (if 0xFFFFFFFF, it means that all the selected sectors have been erased)Program: Address which was selected for data program
Return values	• none:
17.2.11	HAL_FLASH_OperationErrorCallback
Function Name	void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)
Function Description	FLASH operation error interrupt callback.
Parameters	• ReturnValue: The value saved in this parameter depends on the ongoing procedurePageess Erase: Sector number which returned an errorProgram: Address which was selected for data program
Return values	• none:
17.2.12	HAL_FLASH_Unlock
Function Name	HAL_StatusTypeDef HAL_FLASH_Unlock (void)
Function Description	Unlock the FLASH control register access.
Return values	• HAL_StatusTypeDef: HAL Status
17.2.13	HAL_FLASH_Lock
Function Name	HAL_StatusTypeDef HAL_FLASH_Lock (void)
Function Description	Locks the FLASH control register access.
Return values	• HAL_StatusTypeDef: HAL Status
17.2.14	HAL_FLASH_OB_Unlock

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)
Function Description	Unlock the FLASH Option Control Registers access.
Return values	• HAL_StatusTypeDef: HAL Status

17.2.15 HAL_FLASH_OB_Lock

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)
Function Description	Lock the FLASH Option Control Registers access.
Return values	• HAL_StatusTypeDef: HAL Status

17.2.16 HAL_FLASH_OB_Launch

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)
Function Description	Launch the option byte loading.
Return values	• HAL_StatusTypeDef: HAL Status

17.2.17 HAL_FLASH_GetError

Function Name	uint32_t HAL_FLASH_GetError (void)
Function Description	Get the specific FLASH error flag.
Return values	• uint32_t: The returned value can be a mixed of : HAL_FLASH_ERROR_RD: FLASH Read Protection error flag (PCROP) HAL_FLASH_ERROR_SIZE: FLASH Programming Parallelism error flag HAL_FLASH_ERROR_PGA: FLASH Programming Alignment error flag HAL_FLASH_ERROR_WRP: FLASH Write protected error flag HAL_FLASH_ERROR_OPTV: FLASH Option valid error flag HAL_FLASH_ERROR_FWWERR: FLASH Write or Erase operation aborted HAL_FLASH_ERROR_NOTZERO: FLASH Write operation is done in a not-erased region

17.3 FLASH Firmware driver defines

17.3.1 FLASH

Flash Error Code

HAL_FLASH_ERROR_NONE
 HAL_FLASH_ERROR_RD
 HAL_FLASH_ERROR_SIZE
 HAL_FLASH_ERROR_PGA
 HAL_FLASH_ERROR_WRP
 HAL_FLASH_ERROR_OPTV
 HAL_FLASH_ERROR_FWWERR
 HAL_FLASH_ERROR_NOTZERO

FLASH Public Constants

FLASH_SIZE

FLASH_PAGE_SIZE

FLASH Flags

FLASH_FLAG_BSY	FLASH Busy flag
FLASH_FLAG_EOP	FLASH End of Programming flag
FLASH_FLAG_ENDHV	FLASH End of High Voltage flag
FLASH_FLAG_READY	FLASH Ready flag after low power mode
FLASH_FLAG_WRPERR	FLASH Write protected error flag
FLASH_FLAG_PGAERR	FLASH Programming Alignment error flag
FLASH_FLAG_SIZERR	FLASH Size error flag
FLASH_FLAG_OPTVERR	FLASH Option Validity error flag (not valid with STM32L031xx/STM32L041xx)
FLASH_FLAG_RDERR	FLASH Read protected error flag
FLASH_FLAG_FWWERR	FLASH Write or Erase operation aborted
FLASH_FLAG_NOTZEROERR	FLASH Read protected error flag

FLASH Interrupts

FLASH_IT_EOP	End of programming interrupt source
FLASH_IT_ERR	Error interrupt source

FLASH Keys

FLASH_PDKEY1	Flash power down key1
FLASH_PDKEY2	Flash power down key2: used with FLASH_PDKEY1 to unlock the RUN_PD bit in FLASH_ACR
FLASH_PEKEY1	Flash program erase key1
FLASH_PEKEY2	Flash program erase key: used with FLASH_PEKEY2 to unlock the write access to the FLASH_PECR register and data EEPROM
FLASH_PRGKEY1	Flash program memory key1
FLASH_PRGKEY2	Flash program memory key2: used with FLASH_PRGKEY2 to unlock the program memory
FLASH_OPTKEY1	Flash option key1
FLASH_OPTKEY2	Flash option key2: used with FLASH_OPTKEY1 to unlock the write access to the option byte block

FLASH Latency

FLASH_LATENCY_0	FLASH Zero Latency cycle
FLASH_LATENCY_1	FLASH One Latency cycle

Macros to handle FLASH interrupts

__HAL_FLASH_ENABLE_IT	Description:
	Enable the specified FLASH interrupt.
	Parameters:

- `__INTERRUPT__` : FLASH interrupt This parameter can be any combination of the following values:
 - `FLASH_IT_EOP`: End of FLASH Operation Interrupt
 - `FLASH_IT_ERR`: Error Interrupt

Return value:

- none

`__HAL_FLASH_DISABLE_IT`**Description:**

- Disable the specified FLASH interrupt.

Parameters:

- `__INTERRUPT__` : FLASH interrupt This parameter can be any combination of the following values:
 - `FLASH_IT_EOP`: End of FLASH Operation Interrupt
 - `FLASH_IT_ERR`: Error Interrupt

Return value:

- none

`__HAL_FLASH_GET_FLAG`**Description:**

- Get the specified FLASH flag status.

Parameters:

- `__FLAG__` : specifies the FLASH flag to check. This parameter can be one of the following values:
 - `FLASH_FLAG_BSY` : FLASH Busy flag
 - `FLASH_FLAG_EOP`: FLASH End of Operation flag
 - `FLASH_FLAG_READY`: FLASH Ready flag after low power mode
 - `FLASH_FLAG_ENDHV`: FLASH End of high voltage flag
 - `FLASH_FLAG_WRPERR`: FLASH Write protected error flag
 - `FLASH_FLAG_PGAERR`: FLASH Programming Alignment error flag (not valid with STM32L031xx/STM32L041xx)
 - `FLASH_FLAG_SIZERR`: FLASH Size error flag
 - `FLASH_FLAG_OPTVERR`: FLASH Option validity error flag (not valid with STM32L031xx/STM32L041xx)
 - `FLASH_FLAG_RDERR`: FLASH Read protected error flag
 - `FLASH_FLAG_FWWERR`: FLASH Fetch While Write Error flag
 - `FLASH_FLAG_NOTZEROERR`: Not Zero area 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_EOP: FLASH End of Operation flag
 - FLASH_FLAG_WRPERR: FLASH Write protected error flag
 - FLASH_FLAG_PGAERR: FLASH Programming Alignment error flag (not valid with STM32L031xx/STM32L041xx)
 - FLASH_FLAG_SIZERR: FLASH size error flag
 - FLASH_FLAG_OPTVERR: FLASH Option validity error flag (not valid with STM32L031xx/STM32L041xx)
 - FLASH_FLAG_RDERR: FLASH Read protected error flag
 - FLASH_FLAG_FWWERR: FLASH Fetch While Write Error flag
 - FLASH_FLAG_NOTZEROERR: Not Zero area error flag

Return value:

- None

FLASH Type Program

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

18 HAL FLASH Extension Driver

18.1 FLASHEx Firmware driver registers structures

18.1.1 FLASH_OBProgramInitTypeDef

Data Fields

- *uint32_t* **OptionType**
- *uint32_t* **WRPState**
- *uint32_t* **WRPSector**
- *uint32_t* **WRPSector2**
- *uint8_t* **RDPLLevel**
- *uint8_t* **BORLevel**
- *uint8_t* **USERConfig**
- *uint8_t* **BOOTBit1Config**

Field Documentation

- *uint32_t* **FLASH_OBProgramInitTypeDef::OptionType**
OptionType: Option byte to be configured. This parameter can be a value of [FLASHEx_Option_Type](#)
- *uint32_t* **FLASH_OBProgramInitTypeDef::WRPState**
WRPState: Write protection activation or deactivation. This parameter can be a value of [FLASHEx_WRP_State](#)
- *uint32_t* **FLASH_OBProgramInitTypeDef::WRPSector**
WRPSector: This bitfield specifies the sector (s) which are write protected. This parameter can be a combination of [FLASHEx_Option_Bytes_Write_Protection](#)
- *uint32_t* **FLASH_OBProgramInitTypeDef::WRPSector2**
WRPSector2 : This bitfield specifies the sector(s) upper Sector31 which are write protected. This parameter can be a combination of [FLASHEx_Option_Bytes_Write_Protection2](#)
- *uint8_t* **FLASH_OBProgramInitTypeDef::RDPLLevel**
RDPLLevel: Set the read protection level. This parameter can be a value of [FLASHEx_Option_Bytes_Read_Protection](#)
- *uint8_t* **FLASH_OBProgramInitTypeDef::BORLevel**
BORLevel: Set the BOR Level. This parameter can be a value of [FLASHEx_Option_Bytes_BOR_Level](#)
- *uint8_t* **FLASH_OBProgramInitTypeDef::USERConfig**
USERConfig: Program the FLASH User Option Byte: IWDG_SW / RST_STOP / RST_STDBY. This parameter can be a combination of [FLASHEx_Option_Bytes_IWatchdog](#), [FLASHEx_Option_Bytes_nRST_STOP](#) and [FLASHEx_Option_Bytes_nRST_STDBY](#)
- *uint8_t* **FLASH_OBProgramInitTypeDef::BOOTBit1Config**
BOOT1Config: Together with input pad Boot0, this bit selects the boot source, flash, ram or system memory This parameter can be a value of [FLASHEx_Option_Bytes_BOOTBit1](#)

18.1.2 FLASH_AdvOBProgramInitTypeDef

Data Fields

- *uint32_t OptionType*
- *uint8_t PCROPState*
- *uint32_t PCROPSector*
- *uint32_t PCROPSector2*
- *uint8_t BootConfig*

Field Documentation

- *uint32_t FLASH_AdvOBProgramInitTypeDef::OptionType*
OptionType: Option byte to be configured for extension . This parameter can be a value of [FLASHEx_OptionAdv_Type](#)
- *uint8_t FLASH_AdvOBProgramInitTypeDef::PCROPState*
PCROPState: PCROP activation or deactivation. This parameter can be a value of [FLASHEx_PCROP_State](#)
- *uint32_t FLASH_AdvOBProgramInitTypeDef::PCROPSector*
PCROPSector : This bitfield specifies the sector(s) which are read/write protected. This parameter can be a combination of [FLASHEx_Option_Bytes_PC_ReadWrite_Protection](#)
- *uint32_t FLASH_AdvOBProgramInitTypeDef::PCROPSector2*
PCROPSector : This bitfield specifies the sector(s) upper Sector31 which are read/write protected. This parameter can be a combination of [FLASHEx_Option_Bytes_PC_ReadWrite_Protection2](#)
- *uint8_t FLASH_AdvOBProgramInitTypeDef::BootConfig*
BootConfig: specifies Option bytes for boot config. This parameter can be a value of [FLASHEx_Option_Bytes_BOOT_BANK](#)

18.2 FLASHEx Firmware driver API description

18.2.1 Flash peripheral Extended features

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

- Erase functions
- DATA_EEPROM memory management
- BOOT option bit configuration
- PCROP protection for all sectors

18.2.2 How to use this driver

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

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

18.2.3 FLASH Erasing Programming functions

The FLASH Memory Erasing functions, includes the following functions:

- HAL_FLASHEx_Erase: return only when erase has been done
- HAL_FLASHEx_Erase_IT: end of erase is done when HAL_FLASH_EndOfOperationCallback is called with parameter 0xFFFFFFFF

Any operation of erase should follow these steps:

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

This section contains the following APIs:

- [HAL_FLASHEx_Erase\(\)](#)
- [HAL_FLASHEx_Erase_IT\(\)](#)

18.2.4 Option Bytes Programming functions

Any operation of erase or program should follow these steps:

1. Call the HAL_FLASH_OB_Unlock() function to enable the Flash option control register access.
2. Call following function to program the desired option bytes.
 - HAL_FLASHEx_OBProgram: - To Enable/Disable the desired sector write protection. - To set the desired read Protection Level. - To configure the user option Bytes: IWDG, STOP and the Standby. - To Set the BOR level.
3. Once all needed option bytes to be programmed are correctly written, call the HAL_FLASH_OB_Launch(void) function to launch the Option Bytes programming process.
4. Call the HAL_FLASH_OB_Lock() to disable the Flash option control register access (recommended to protect the option Bytes against possible unwanted operations).

Proprietary code Read Out Protection (PcROP):

1. The PcROP sector is selected by using the same option bytes as the Write protection (nWRPi bits). As a result, these 2 options are exclusive each other.
2. In order to activate the PcROP (change the function of the nWRPi option bits), the WPRMOD option bit must be activated.
3. The active value of nWRPi bits is inverted when PCROP mode is active, this means: if WPRMOD = 1 and nWRPi = 1 (default value), then the user sector "i" is read/write protected.
4. To activate PCROP mode for Flash sector(s), you need to call the following function:
 - HAL_FLASHEx_AdvOBProgram in selecting sectors to be read/write protected
 - HAL_FLASHEx_OB_SelectPCROP to enable the read/write protection

This section contains the following APIs:

- [HAL_FLASHEx_OBProgram\(\)](#)
- [HAL_FLASHEx_OBGetConfig\(\)](#)
- [HAL_FLASHEx_AdvOBProgram\(\)](#)
- [HAL_FLASHEx_AdvOBGetConfig\(\)](#)
- [HAL_FLASHEx_OB_SelectPCROP\(\)](#)
- [HAL_FLASHEx_OB_DeSelectPCROP\(\)](#)

18.2.5 DATA EEPROM Programming functions

The FLASH_DATAEEPROM Programming_Functions, includes the following functions:

- HAL_FLASHEx_DATAEEPROM_Unlock(void);
- HAL_FLASHEx_DATAEEPROM_Lock(void);
- HAL_FLASHEx_DATAEEPROM_Erase(uint32_t Address)
- HAL_FLASHEx_DATAEEPROM_Program(uint32_t TypeProgram, uint32_t Address, uint32_t Data)

Any operation of erase or program should follow these steps:

1. Call the HAL_FLASHEx_DATAEEPROM_Unlock() function to enable the data EEPROM access and Flash program erase control register access.
2. Call the desired function to erase or program data.
3. Call the HAL_FLASHEx_DATAEEPROM_Lock() to disable the data EEPROM access and Flash program erase control register access(recommended to protect the DATA_EEPROM against possible unwanted operation).

This section contains the following APIs:

- [HAL_FLASHEx_DATAEEPROM_Unlock\(\)](#)
- [HAL_FLASHEx_DATAEEPROM_Lock\(\)](#)
- [HAL_FLASHEx_DATAEEPROM_Erase\(\)](#)
- [HAL_FLASHEx_DATAEEPROM_Program\(\)](#)
- [HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram\(\)](#)
- [HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram\(\)](#)

18.2.6 HAL_FLASHEx_Erase

Function Name	HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef *pEraseInit, uint32_t *PageError)
Function Description	Erase the specified FLASH memory Pages.
Parameters	• pEraseInit: pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing. • PageError: pointer to variable that contains the configuration information on faulty sector in case of error (0xFFFFFFFF means that all the sectors have been correctly erased)
Return values	• HAL_StatusTypeDef: HAL Status
Notes	• To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation)

18.2.7 HAL_FLASHEx_Erase_IT

Function Name	HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef *pEraseInit)
Function Description	Perform a page erase of the specified FLASH memory pages with interrupt enabled.
Parameters	• pEraseInit: pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
Return values	• HAL_StatusTypeDef: HAL Status

Notes	To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation). End of erase is done when HAL_FLASH_EndOfOperationCallback is called with parameter 0xFFFFFFFF

18.2.8 HAL_FLASHEx_OBProgram

Function Name	HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)
Function Description	Program option bytes.
Parameters	pOBInit: pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	HAL_StatusTypeDef: HAL Status

18.2.9 HAL_FLASHEx_OBGetConfig

Function Name	void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)
Function Description	Get the Option byte configuration.
Parameters	pOBInit: pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	None:

18.2.10 HAL_FLASHEx_AdvOBProgram

Function Name	HAL_StatusTypeDef HAL_FLASHEx_AdvOBProgram (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)
Function Description	Program option bytes.
Parameters	pAdvOBInit: pointer to an FLASH_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.
Return values	HAL_StatusTypeDef: HAL Status

18.2.11 HAL_FLASHEx_AdvOBGetConfig

Function Name	void HAL_FLASHEx_AdvOBGetConfig (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)
Function Description	Get the OBEX byte configuration.
Parameters	pAdvOBInit: pointer to an FLASH_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.
Return values	None:

18.2.12 HAL_FLASHEx_OB_SelectPCROP

Function Name	HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP (void)
Function Description	Select the Protection Mode (WPRMOD).
Return values	• HAL: status
Notes	• Once WPRMOD bit is active, unprotection of a protected sector is not possible • Read a protected sector will set RDERR Flag and write a protected sector will set WRPERR Flag

18.2.13 HAL_FLASHEx_OB_DeSelectPCROP

Function Name	HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP (void)
Function Description	Deselect the Protection Mode (WPRMOD).
Return values	• HAL: status
Notes	• Once WPRMOD bit is active, unprotection of a protected sector is not possible • Read a protected sector will set RDERR Flag and write a protected sector will set WRPERR Flag

18.2.14 HAL_FLASHEx_DATAEEPROM_Unlock

Function Name	HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Unlock (void)
Function Description	Unlocks the data memory and FLASH_PECR register access.
Return values	• HAL_StatusTypeDef: HAL Status

18.2.15 HAL_FLASHEx_DATAEEPROM_Lock

Function Name	HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Lock (void)
Function Description	Locks the Data memory and FLASH_PECR register access.
Return values	• HAL_StatusTypeDef: HAL Status

18.2.16 HAL_FLASHEx_DATAEEPROM_Erase

Function Name	HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Erase (uint32_t Address)
Function Description	Erase a word in data memory.
Parameters	• Address: specifies the address to be erased.
Return values	• HAL: status
Notes	• To correctly run this function, the HAL_FLASHEx_DATAEEPROM_Unlock() function must be called before. Call the HAL_FLASHEx_DATAEEPROM_Lock() to the data EEPROM access and Flash program erase control register access(recommended to protect the DATA_EEPROM against possible unwanted operation).

18.2.17 HAL_FLASHEx_DATAEEPROM_Program

Function Name	HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Program (uint32_t TypeProgram, uint32_t Address, uint32_t Data)
Function Description	Program word at a specified address.
Parameters	• TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program Data • Address: specifies the address to be programmed. • Data: specifies the data to be programmed
Return values	• HAL_StatusTypeDef: HAL Status

18.2.18 HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram

Function Name	void HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram (void)
Function Description	Enable DATA EEPROM fixed Time programming (2*Tprog).
Return values	• None:

18.2.19 HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram

Function Name	void HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram (void)
Function Description	Disables DATA EEPROM fixed Time programming (2*Tprog).
Return values	• None:

18.3 FLASHEx Firmware driver defines**18.3.1 FLASHEx***FLASHEx Address*

FLASH_NBPAGES_MAX

FLASHEx Exported Macros

__HAL_FLASH_SET_LATENCY

Description:

- Set the FLASH Latency.

Parameters:

- **__LATENCY__:** FLASH Latency This parameter can be one of the following values:
 - FLASH_LATENCY_0: FLASH Zero Latency cycle
 - FLASH_LATENCY_1: FLASH One Latency cycle

Return value:

__HAL_FLASH_GET_LATENCY

- none

Description:

- Get the FLASH Latency.

Return value:

- FLASH: Latency This parameter can be one of the following values:
 - FLASH_LATENCY_0: FLASH Zero Latency cycle
 - FLASH_LATENCY_1: FLASH One Latency cycle

__HAL_FLASH_PREFETCH_BUFFER_ENABLE

Description:

- Enable/Disable the FLASH prefetch buffer.

Return value:

- none

__HAL_FLASH_PREFETCH_BUFFER_DISABLE

__HAL_FLASH_BUFFER_CACHE_ENABLE

Description:

- Enable/Disable the FLASH Buffer cache.

Return value:

- none

__HAL_FLASH_BUFFER_CACHE_DISABLE

__HAL_FLASH_PREREAD_BUFFER_ENABLE

Description:

- Enable/Disable the FLASH preread buffer.

Return value:

- none

__HAL_FLASH_PREREAD_BUFFER_DISABLE

__HAL_FLASH_SLEEP_POWERDOWN_ENABLE

Description:

- Enable/Disable the FLASH power down during Sleep mode.

Return value:

- none

__HAL_FLASH_SLEEP_POWERDOWN_DISABLE

__HAL_FLASH_POWER_DOWN_ENABLE

Notes:

- Writing this bit to 0 this bit, automatically the keys are loss and a new unlock sequence is necessary to re-write it to 1.

__HAL_FLASH_POWER_DOWN_DISABLE

Notes:

- Writing this bit to 0 this bit, automatically the keys are loss and a new unlock sequence is necessary to re-write it to 1.

FLASH Option Byte

OPTIONBYTE_PCROP PCROP option byte configuration

OPTIONBYTE_BOOTCONFIG BOOTConfig option byte configuration, boot from bank 2

FLASH Option Bytes BOOT Bit1 Setup

OB_BOOT_BIT1_RESET BOOT Bit 1 Reset

OB_BOOT_BIT1_SET BOOT Bit 1 Set

FLASH Option Bytes BOOT BANK

OB_BOOT_BANK1 At startup, if boot pin 0 and BOOT1 bit are set in boot from user Flash position and this parameter is selected the device will boot from Bank 1 (Default)

OB_BOOT_BANK2 At startup, if boot pin 0 and BOOT1 bit are set in boot from user Flash position and this parameter is selected the device will boot from Bank 2

FLASH Option Bytes BOR Level

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

OB_BOR_LEVEL1 BOR Reset threshold levels for 1.7V - 1.8V VDD power supply

OB_BOR_LEVEL2 BOR Reset threshold levels for 1.9V - 2.0V VDD power supply

OB_BOR_LEVEL3 BOR Reset threshold levels for 2.3V - 2.4V VDD power supply

OB_BOR_LEVEL4 BOR Reset threshold levels for 2.55V - 2.65V VDD power supply

OB_BOR_LEVEL5 BOR Reset threshold levels for 2.8V - 2.9V VDD power supply

FLASH Option Bytes IWatchdog

OB_IWDG_SW Software WDG selected

OB_IWDG_HW Hardware WDG selected

FLASH Option Bytes nRST_STDBY

OB_STDBY_NORST No reset generated when entering in STANDBY

OB_STDBY_RST Reset generated when entering in STANDBY

FLASHEx Option Bytes nRST_STOP

OB_STOP_NORST No reset generated when entering in STOP

OB_STOP_RST Reset generated when entering in STOP

FLASH Option Bytes PC Read/Write Protection

OB_PCROP_Pages0to31

OB_PCROP_Pages32to63

OB_PCROP_Pages64to95
OB_PCROP_Pages96to127
OB_PCROP_Pages128to159
OB_PCROP_Pages160to191
OB_PCROP_Pages192to223
OB_PCROP_Pages224to255
OB_PCROP_Pages256to287
OB_PCROP_Pages288to319
OB_PCROP_Pages320to351
OB_PCROP_Pages352to383
OB_PCROP_Pages384to415
OB_PCROP_Pages416to447
OB_PCROP_Pages448to479
OB_PCROP_Pages480to511
OB_PCROP_Pages512to543
OB_PCROP_Pages544to575
OB_PCROP_Pages576to607
OB_PCROP_Pages608to639
OB_PCROP_Pages640to671
OB_PCROP_Pages672to703
OB_PCROP_Pages704to735
OB_PCROP_Pages736to767
OB_PCROP_Pages768to799
OB_PCROP_Pages800to831
OB_PCROP_Pages832to863
OB_PCROP_Pages864to895
OB_PCROP_Pages896to927
OB_PCROP_Pages928to959
OB_PCROP_Pages960to991
OB_PCROP_Pages992to1023

OB_PCROP_AllPages PC Read/Write protection of all Sectors

FLASH Option Bytes PC Read/Write Protection (Sector 2)

OB_PCROP2_Pages1024to1055
OB_PCROP2_Pages1056to1087
OB_PCROP2_Pages1088to1119
OB_PCROP2_Pages1120to1151

OB_PCROP2_Pages1152to1183

OB_PCROP2_Pages1184to1215

OB_PCROP2_Pages1216to1247

OB_PCROP2_Pages1248to1279

OB_PCROP2_Pages1280to1311

OB_PCROP2_Pages1312to1343

OB_PCROP2_Pages1344to1375

OB_PCROP2_Pages1376to1407

OB_PCROP2_Pages1408to1439

OB_PCROP2_Pages1440to1471

OB_PCROP2_Pages1472to1503

OB_PCROP2_Pages1504to1535

OB_PCROP2_AllPages PC Read/Write protection of all Sectors PCROP2

FLASH Option Bytes Write Mask

WRP_MASK_LOW

WRP_MASK_HIGH

FLASH Option Bytes Read Protection

OB_RDP_LEVEL_0

OB_RDP_LEVEL_1

OB_RDP_LEVEL_2

FLASH Option Bytes Write ProtectionP

OB_WRP_Pages0to31

OB_WRP_Pages32to63

OB_WRP_Pages64to95

OB_WRP_Pages96to127

OB_WRP_Pages128to159

OB_WRP_Pages160to191

OB_WRP_Pages192to223

OB_WRP_Pages224to255

OB_WRP_Pages256to287

OB_WRP_Pages288to319

OB_WRP_Pages320to351

OB_WRP_Pages352to383

OB_WRP_Pages384to415

OB_WRP_Pages416to447

OB_WRP_Pages448to479

OB_WRP_Pages480to511

OB_WRP_Pages512to543

OB_WRP_Pages544to575

OB_WRP_Pages576to607

OB_WRP_Pages608to639

OB_WRP_Pages640to671

OB_WRP_Pages672to703

OB_WRP_Pages704to735

OB_WRP_Pages736to767

OB_WRP_Pages768to799

OB_WRP_Pages800to831

OB_WRP_Pages832to863

OB_WRP_Pages864to895

OB_WRP_Pages896to927

OB_WRP_Pages928to959

OB_WRP_Pages960to991

OB_WRP_Pages992to1023

OB_WRP_AllPages Write protection of all Sectors

FLASH Option Bytes Write Protection

OB_WRP2_Pages1024to1055

OB_WRP2_Pages1056to1087

OB_WRP2_Pages1088to1119

OB_WRP2_Pages1120to1151

OB_WRP2_Pages1152to1183

OB_WRP2_Pages1184to1215

OB_WRP2_Pages1216to1247

OB_WRP2_Pages1248to1279

OB_WRP2_Pages1280to1311

OB_WRP2_Pages1312to1343

OB_WRP2_Pages1344to1375

OB_WRP2_Pages1376to1407

OB_WRP2_Pages1408to1439

OB_WRP2_Pages1440to1471

OB_WRP2_Pages1472to1503

OB_WRP2_Pages1504to1535

OB_WRP2_AllPages Write protection of all Sectors WRP2

FLASH Option Type

OPTIONBYTE_WRP	WRP option byte configuration
OPTIONBYTE_RDP	RDP option byte configuration
OPTIONBYTE_USER	USER option byte configuration
OPTIONBYTE_BOR	BOR option byte configuration
OPTIONBYTE_BOOT_BIT1	BOOT PIN1 option byte configuration

FLASH PCROP State

OB_PCROP_STATE_DISABLE	Disable PCROP
OB_PCROP_STATE_ENABLE	Enable PCROP

FLASH Type Erase

FLASH_TYPEERASE_PAGES	Page erase only

FLASH Type Program Data

FLASH_TYPEPROGRAMDATA_BYTE	Program byte (8-bit) at a specified address.
FLASH_TYPEPROGRAMDATA_HALFWORD	Program a half-word (16-bit) at a specified address.
FLASH_TYPEPROGRAMDATA_WORD	Program a word (32-bit) at a specified address.
FLASH_TYPEPROGRAM_BYTE	
FLASH_TYPEPROGRAM_HALFWORD	

FLASH WRP State

OB_WRPSTATE_DISABLE	Disable the write protection of the desired sectors
OB_WRPSTATE_ENABLE	Enable the write protection of the desired sectors

19 HAL FLASH__RAMFUNC Generic Driver

19.1 FLASH__RAMFUNC Firmware driver API description

19.1.1 HAL_FLASHEx_EnableRunPowerDown

Function Name	<code>__RAM_FUNC HAL_FLASHEx_EnableRunPowerDown (void)</code>
Function Description	Enable the power down mode during RUN mode.
Return values	• HAL: Status
Notes	• This function can be used only when the user code is running from Internal SRAM.

19.1.2 HAL_FLASHEx_DisableRunPowerDown

Function Name	<code>__RAM_FUNC HAL_FLASHEx_DisableRunPowerDown (void)</code>
Function Description	Disable the power down mode during RUN mode.
Return values	• HAL: Status
Notes	• This function can be used only when the user code is running from Internal SRAM.

19.1.3 HAL_FLASHEx_HalfPageProgram

Function Name	<code>__RAM_FUNC HAL_FLASHEx_HalfPageProgram (uint32_t Address, uint32_t * pBuffer)</code>
Function Description	Program a half page in program memory.
Parameters	• Address: specifies the address to be written. • pBuffer: pointer to the buffer containing the data to be written to the half page.
Return values	• HAL: Status: The returned value can be: HAL_ERROR, HAL_OK or HAL_TIMEOUT.
Notes	• To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation) • Half page write is possible only from SRAM. • A half page is written to the program memory only if the first address to load is the start address of a half page (multiple of 64 bytes) and the 15 remaining words to load are in the same half page. • During the Program memory half page write all read operations are forbidden (this includes DMA read operations and debugger read operations such as breakpoints, periodic updates, etc.). • If a PGAERR is set during a Program memory half page write, the complete write operation is aborted. Software should then reset the FPRG and PROG/DATA bits and restart the write

operation from the beginning.

19.1.4 HAL_FLASHRAM_GetError

Function Name	<code>__RAM_FUNC HAL_FLASHRAM_GetError (uint32_t * error)</code>
Function Description	Get the specific FLASH errors flag.
Parameters	error: pointer is the error value. It can be a mixed of : HAL_FLASH_ERROR_RD: FLASH Read Protection error flag (PCROP) HAL_FLASH_ERROR_SIZE: FLASH Programming Parallelism error flag HAL_FLASH_ERROR_PGA: FLASH Programming Alignment error flag HAL_FLASH_ERROR_WRP: FLASH Write protected error flag HAL_FLASH_ERROR_OPTV: FLASH Option valid error flag HAL_FLASH_ERROR_FWWERR: FLASH Write or Erase operation aborted HAL_FLASH_ERROR_NOTZERO: FLASH Write operation is done in a not-erased region
Return values	HAL: Status

19.1.5 HAL_FLASHEx_EraseParallelPage

Function Name	<code>__RAM_FUNC HAL_FLASHEx_EraseParallelPage (uint32_t Page_Address1, uint32_t Page_Address2)</code>
Function Description	Erases a specified 2 pages in program memory in parallel.
Parameters	Page_Address1: The page address in program memory to be erased in the first Bank (BANK1). This parameter should be between FLASH_BASE and FLASH_BANK1_END. Page_Address2: The page address in program memory to be erased in the second Bank (BANK2). This parameter should be between FLASH_BANK2_BASE and FLASH_BANK2_END.
Return values	HAL: Status: The returned value can be: HAL_ERROR, HAL_OK or HAL_TIMEOUT.
Notes	- This function can be used only for STM32L07xxx/STM32L08xxx devices. To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation). - A Page is erased in the Program memory only if the address to load is the start address of a page (multiple of 128 bytes).

19.1.6 HAL_FLASHEx_ProgramParallelHalfPage

Function Name	<code>__RAM_FUNC HAL_FLASHEx_ProgramParallelHalfPage (uint32_t Address1, uint32_t * pBuffer1, uint32_t Address2, uint32_t * pBuffer2)</code>
Function Description	Programs 2 half pages in program memory in parallel.
Parameters	Address1: specifies the first address to be written in the first bank (BANK1). This parameter should be between FLASH_BASE and (FLASH_BANK1_END -

Return values

Notes

- FLASH_PAGE_SIZE).
- **pBuffer1:** pointer to the buffer containing the data to be written to the first half page in the first bank.
- **Address2:** specifies the second address to be written in the second bank (BANK2). This parameter should be between FLASH_BANK2_BASE and (FLASH_BANK2_END - FLASH_PAGE_SIZE).
- **pBuffer2:** pointer to the buffer containing the data to be written to the second half page in the second bank.
- **HAL:** Status: The returned value can be: HAL_ERROR, HAL_OK or HAL_TIMEOUT.
- This function can be used only for STM32L07xxx/STM32L08xxx devices.
- To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation).
- Half page write is possible only from SRAM.
- A half page is written to the program memory only if the first address to load is the start address of a half page (multiple of 64 bytes) and the 15 remaining words to load are in the same half page.
- During the Program memory half page write all read operations are forbidden (this includes DMA read operations and debugger read operations such as breakpoints, periodic updates, etc.).
- If a PGAERR is set during a Program memory half page write, the complete write operation is aborted. Software should then reset the FPRG and PROG/DATA bits and restart the write operation from the beginning.

20 HAL GPIO Generic Driver

20.1 GPIO Firmware driver registers structures

20.1.1 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 a combination of [GPIO_pins_define](#)
- *uint32_t GPIO_InitTypeDef::Mode*
Specifies the operating mode for the selected pins. This parameter can be a value of [GPIO_mode_define](#)
- *uint32_t GPIO_InitTypeDef::Pull*
Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of [GPIO_pull_define](#)
- *uint32_t GPIO_InitTypeDef::Speed*
Specifies the speed for the selected pins. This parameter can be a value of [GPIO_speed_define](#)
- *uint32_t GPIO_InitTypeDef::Alternate*
Peripheral to be connected to the selected pins. This parameter can be a value of [GPIOEx_Alternate_function_selection](#)

20.2 GPIO Firmware driver API description

20.2.1 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.

20.2.2 How to use this driver

1. Enable the GPIO IOPORT 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. This function is also to be used when unconfiguring pin which was used as an external interrupt or in event mode. That is the only way to reset the 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 PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

20.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- [HAL_GPIO_Init\(\)](#)
- [HAL_GPIO_DeInit\(\)](#)

20.2.4 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\(\)](#)

20.2.5 HAL_GPIO_Init

Function Name	void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_Init.
Parameters	• GPIOx: where x can be (A..E and H) to select the GPIO peripheral for STM32L0XX family devices. Note that GPIOE is not available on all devices. • GPIO_Init: pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	• None:

20.2.6 HAL_GPIO_DeInit

Function Name	void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)
Function Description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	• GPIOx: where x can be (A..E and H) to select the GPIO peripheral for STM32L0XX family devices. Note that GPIOE is not available on all devices. • GPIO_Pin: specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). All port bits are not necessarily available on all GPIOs.
Return values	• None:

20.2.7 HAL_GPIO_ReadPin

Function Name	GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
Function Description	Reads the specified input port pin.
Parameters	• GPIOx: where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family devices. Note that GPIOE is not available on all devices. • GPIO_Pin: specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15). All port bits are not

necessarily available on all GPIOs.

Return values

- **The:** input port pin value.

20.2.8 HAL_GPIO_WritePin

Function Name **void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)**

Function Description Sets or clears the selected data port bit.

Parameters

- **GPIOx:** where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family devices. Note that GPIOE is not available on all devices.
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). All port bits are not necessarily available on all GPIOs.
- **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: GPIO_PIN_RESET: to clear the port pin
GPIO_PIN_SET: to set the port pin

Return values

- **None:**

Notes

- This function uses GPIOx_BSRR register to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

20.2.9 HAL_GPIO_TogglePin

Function Name **void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**

Function Description Toggles the specified GPIO pins.

Parameters

- **GPIOx:** Where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family devices. Note that GPIOE is not available on all devices. All port bits are not necessarily available on all GPIOs.
- **GPIO_Pin:** Specifies the pins to be toggled.

Return values

- **None:**

20.2.10 HAL_GPIO_LockPin

Function Name **HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**

Function Description Locks GPIO Pins configuration registers.

Parameters

- **GPIOx:** where x can be (A..E and H) to select the GPIO peripheral for STM32L0xx family. Note that GPIOE is not available on all devices.
- **GPIO_Pin:** specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15). All port bits are not necessarily available on all GPIOs.

Return values

- **None:**

Notes

- The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFR1 and GPIOx_AFR2.
- The configuration of the locked GPIO pins can no longer be modified until the next reset.

20.2.11 HAL_GPIO_EXTI_IRQHandler

Function Name	void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)
Function Description	This function handles EXTI interrupt request.
Parameters	• GPIO_Pin: Specifies the pins connected to the EXTI line.
Return values	• None:

20.2.12 HAL_GPIO_EXTI_Callback

Function Name	void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)
Function Description	EXTI line detection callbacks.
Parameters	• GPIO_Pin: Specifies the pins connected to the EXTI line.
Return values	• None:

20.3 GPIO Firmware driver defines**20.3.1 GPIO****GPIO Exported Constants**

GPIO_PIN_MASK
IS_GPIO_PIN
IS_GPIO_MODE
IS_GPIO_SPEED
IS_GPIO_PULL

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_FLAG

Description:

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

Parameters:

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

Return value:

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

__HAL_GPIO_EXTI_CLEAR_FLAG

Description:

- Clears the EXTI's line pending flags.

`__HAL_GPIO_EXTI_GET_IT`

Parameters:

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

Return value:

- None

Description:

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

Parameters:

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

Return value:

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

Description:

- Clears the EXTI's line pending bits.

Parameters:

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

Return value:

- None

`__HAL_GPIO_EXTI_CLEAR_IT`

Description:

- Generates a Software interrupt on selected EXTI line.

Parameters:

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

Return value:

- None

`__HAL_GPIO_EXTI_GENERATE_SWIT`

GPIO Exported Types

`IS_GPIO_PIN_ACTION`

Mode definition

`GPIO_MODE_INPUT`

Input Floating Mode

`GPIO_MODE_OUTPUT_PP`

Output Push Pull Mode

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

Pin definition

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

Pull definition

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

Speed definition

GPIO_SPEED_FREQ_LOW	range up to 0.4 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_MEDIUM	range 0.4 MHz to 2 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_HIGH	range 2 MHz to 10 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_VERY_HIGH	range 10 MHz to 35 MHz, please refer to the product datasheet

21 HAL GPIO Extension Driver

21.1 GPIOEx Firmware driver defines

21.1.1 GPIOEx

Alternate function selection

GPIO_AF0_EVENTOUT

GPIO_AF0_TIM21

GPIO_AF0_SPI1

GPIO_AF0_MCO

GPIO_AF0_SWDIO

GPIO_AF0_SWCLK

GPIO_AF0_USART1

GPIO_AF0_SPI2

GPIO_AF0_LPTIM1

GPIO_AF0_TIM22

GPIO_AF0_LPUART1

GPIO_AF0_USART2

GPIO_AF0_TIM2

GPIO_AF0_USB

GPIO_AF1_I2C1

GPIO_AF1_SPI2

GPIO_AF1_TIM21

GPIO_AF1_LCD

GPIO_AF2_TIM2

GPIO_AF2_TIM3

GPIO_AF2_EVENTOUT

GPIO_AF2_LPTIM1

GPIO_AF2_LPUART1

GPIO_AF2_MCO

GPIO_AF2_RTC

GPIO_AF2_SPI2

GPIO_AF2_USART5

GPIO_AF2_SPI1

GPIO_AF2_USB

GPIO_AF3_EVENTOUT

GPIO_AF3_I2C1
GPIO_AF3_TSC
GPIO_AF4_USART2
GPIO_AF4_LPUART1
GPIO_AF4_USART1
GPIO_AF4_EVENTOUT
GPIO_AF4_TIM22
GPIO_AF4_TIM3
GPIO_AF4_I2C1
GPIO_AF5_TIM2
GPIO_AF5_TIM21
GPIO_AF5_TIM22
GPIO_AF5_USART1
GPIO_AF5_SPI2
GPIO_AF5_I2C2
GPIO_AF6_USART4
GPIO_AF6_LPUART1
GPIO_AF6_EVENTOUT
GPIO_AF6_I2C1
GPIO_AF6_I2C2
GPIO_AF6_USART5
GPIO_AF6_TIM21
GPIO_AF7_COMP1
GPIO_AF7_COMP2
GPIO_AF7_I2C3
GPIO_AF7_LPUART1

Pin available

GPIOA_PIN_AVAILABLE
GPIOB_PIN_AVAILABLE
GPIOC_PIN_AVAILABLE
GPIOD_PIN_AVAILABLE
GPIOE_PIN_AVAILABLE
GPIOH_PIN_AVAILABLE

22 HAL I2C Generic Driver

22.1 I2C Firmware driver registers structures

22.1.1 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](#)

22.1.2 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, see I2C_Error_Code

22.2 I2C Firmware driver API description

22.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C_HandleTypeDef handle structure, for example: I2C_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implement the HAL_I2C_MspInit ()API:
 - a. Enable the I2Cx interface clock
 - b. I2C pins configuration
 - Enable the clock for the I2C GPIOs
 - Configure I2C pins as alternate function open-drain
 - c. NVIC configuration if you need to use interrupt process
 - Configure the I2Cx interrupt priority
 - Enable the NVIC I2C IRQ Channel
 - d. DMA Configuration if you need to use DMA process

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

Polling mode IO operation

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

Polling mode IO MEM operation

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

Interrupt mode IO operation

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

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

Interrupt mode IO MEM operation

- Write an amount of data in no-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At MEM end of write transfer HAL_I2C_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback
- Read an amount of data in no-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At MEM end of read transfer HAL_I2C_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

DMA mode IO operation

- Transmit in master mode an amount of data in non blocking mode (DMA) using HAL_I2C_Master_Transmit_DMA()
- At transmission end of transfer HAL_I2C_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode (DMA) using HAL_I2C_Master_Receive_DMA()
- At reception end of transfer HAL_I2C_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode (DMA) using HAL_I2C_Slave_Transmit_DMA()
- At transmission end of transfer HAL_I2C_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode (DMA) using HAL_I2C_Slave_Receive_DMA()
- At reception end of transfer HAL_I2C_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

DMA mode IO MEM operation

- Write an amount of data in no-blocking mode with DMA to a specific memory address using HAL_I2C_Mem_Write_DMA()

- At MEM end of write transfer HAL_I2C_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback
- Read an amount of data in no-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At MEM end of read transfer HAL_I2C_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

I2C HAL driver macros list

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

- __HAL_I2C_ENABLE: Enable the I2C peripheral
- __HAL_I2C_DISABLE: Disable the I2C peripheral
- __HAL_I2C_GET_FLAG : Check 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

22.2.2 Initialization and de-initialization functions

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

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

22.2.3 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\(\)](#)

22.2.4 Peripheral State and Errors 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_I2C_GetState\(\)](#)
- [HAL_I2C_GetError\(\)](#)

22.2.5 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 create the associated handle.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• HAL: status

22.2.6 HAL_I2C_DeInit

Function Name	HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)
Function Description	DeInitializes the I2C peripheral.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• HAL: status

22.2.7 HAL_I2C_MspInit

Function Name	void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)
Function Description	I2C MSP Init.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None:

22.2.8 HAL_I2C_MspDeInit

Function Name	void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)
Function Description	I2C MSP DeInit.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None:

22.2.9 HAL_I2C_Master_Transmit

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits in master mode an amount of data in blocking mode.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	• HAL: status

22.2.10 HAL_I2C_Master_Receive

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receives in master mode an amount of data in blocking mode.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	• HAL: status

22.2.11 HAL_I2C_Slave_Transmit

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits in slave mode an amount of data in blocking mode.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	• HAL: status

22.2.12 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	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

22.2.13 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 no-blocking mode with Interrupt.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

22.2.14 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 no-blocking mode with Interrupt.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

22.2.15 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 no-blocking mode with Interrupt.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

22.2.16 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 no-blocking mode with Interrupt.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	• HAL: status
22.2.17 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 no-blocking mode with DMA.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	• HAL: status
22.2.18 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 no-blocking mode with DMA.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	• HAL: status
22.2.19 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 no-blocking mode with DMA.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • pData: Pointer to data buffer • Size: Amount of data to be sent

Return values

- **HAL:** status

22.2.20 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 no-blocking mode with DMA.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

22.2.21 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

- **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

22.2.22 HAL_I2C_Mem_Read

Function Name `HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)`

Function Description Read an amount of data in blocking mode from a specific memory address.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

22.2.23 HAL_I2C_Mem_Write_IT

Function Name	HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)
Function Description	Write an amount of data in no-blocking mode with Interrupt to a specific memory address.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • MemAddress: Internal memory address • MemAddSize: Size of internal memory address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	• HAL: status

22.2.24 HAL_I2C_Mem_Read_IT

Function Name	HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)
Function Description	Read an amount of data in no-blocking mode with Interrupt from a specific memory address.
Parameters	• 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

22.2.25 HAL_I2C_Mem_Write_DMA

Function Name	HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)
Function Description	Write an amount of data in no-blocking mode with DMA to a specific memory address.
Parameters	• hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for 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

22.2.26 HAL_I2C_Mem_Read_DMA

Function Name `HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)`

Function Description Reads an amount of data in no-blocking mode with DMA from a specific memory address.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for 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

- **HAL:** status

22.2.27 HAL_I2C_IsDeviceReady

Function Name `HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)`

Function Description Checks if target device is ready for communication.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used with Memory devices

22.2.28 HAL_I2C_EV_IRQHandler

Function Name `void HAL_I2C_EV_IRQHandler(I2C_HandleTypeDef * hi2c)`

Function Description This function handles I2C event interrupt request.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

22.2.29 HAL_I2C_ER_IRQHandler

Function Name `void HAL_I2C_ER_IRQHandler(I2C_HandleTypeDef * hi2c)`

Function Description This function handles I2C error interrupt request.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

22.2.30 HAL_I2C_MasterTxCpltCallback

Function Name **void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function Description Master Tx Transfer completed callbacks.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

22.2.31 HAL_I2C_MasterRxCpltCallback

Function Name **void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function Description Master Rx Transfer completed callbacks.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

22.2.32 HAL_I2C_SlaveTxCpltCallback

Function Name **void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function Description Slave Tx Transfer completed callbacks.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

22.2.33 HAL_I2C_SlaveRxCpltCallback

Function Name **void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function Description Slave Rx Transfer completed callbacks.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

22.2.34 HAL_I2C_MemTxCpltCallback

Function Name **void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function Description Memory Tx Transfer completed callbacks.

Parameters

- **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

22.2.35 HAL_I2C_MemRxCpltCallback

Function Name	void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Memory Rx Transfer completed callbacks.
Parameters	hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	None:

22.2.36 HAL_I2C_ErrorCallback

Function Name	void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)
Function Description	I2C error callbacks.
Parameters	hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	None:

22.2.37 HAL_I2C_GetState

Function Name	HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)
Function Description	Returns the I2C state.
Parameters	hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	HAL: state

22.2.38 HAL_I2C_GetError

Function Name	uint32_t HAL_I2C_GetError (I2C_HandleTypeDef * hi2c)
Function Description	Return the I2C error code.
Parameters	hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	I2C: Error Code

22.3 I2C Firmware driver defines**22.3.1 I2C*****I2C addressing mode***

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

I2C dual addressing mode

I2C_DUALADDRESS_DISABLE

I2C_DUALADDRESS_ENABLE

I2C Error Code

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

I2C Exported Macros

`__HAL_I2C_RESET_HANDLE_STATE` **Description:**

- Reset I2C handle state.

Parameters:

- `__HANDLE__`: specifies the I2C Handle.

Return value:

- None

`__HAL_I2C_ENABLE_IT`

Description:

- Enable the specified I2C interrupts.

Parameters:

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

Description:

- Disable the specified I2C interrupts.

Parameters:

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

- I2C_IT_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**Description:**

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

Parameters:

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

Return value:

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

I2C_FLAG_MASK**Description:**

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

Parameters:

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

- Clears 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:

__HAL_I2C_ENABLE

- None

Description:

- Enable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_DISABLE

Description:

- Disable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

I2C Flag definition

I2C_FLAG_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

23 HAL I2C Extension Driver

23.1 I2CEx Firmware driver API description

23.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32L0XX 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

23.1.2 How to use this driver

This driver provides functions to configure Noise Filter

1. Configure I2C Analog noise filter using the function `HAL_I2CEx_ConfigAnalogFilter()`
2. Configure I2C Digital noise filter using the function `HAL_I2CEx_ConfigDigitalFilter()`
3. Configure the enable or disable of I2C Wake Up Mode using the functions : + `HAL_I2CEx_EnableWakeUp()` + `HAL_I2CEx_DisableWakeUp()`
4. Configure the enable or disable of fast mode plus driving capability using the functions : + `HAL_I2CEx_EnableFastModePlus()` + `HAL_I2CEx_DisableFastModePlus()`

23.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:

- [`HAL\_I2CEx\_ConfigAnalogFilter\(\)`](#)
- [`HAL\_I2CEx\_ConfigDigitalFilter\(\)`](#)
- [`HAL\_I2CEx\_EnableWakeUp\(\)`](#)
- [`HAL\_I2CEx\_DisableWakeUp\(\)`](#)
- [`HAL\_I2CEx\_EnableFastModePlus\(\)`](#)
- [`HAL\_I2CEx\_DisableFastModePlus\(\)`](#)

23.1.4 HAL_I2CEx_ConfigAnalogFilter

Function Name	<code>HAL_StatusTypeDef HAL_I2CEx_ConfigAnalogFilter(I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)</code>
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

23.1.5 HAL_I2CEx_ConfigDigitalFilter

Function Name	HAL_StatusTypeDef HAL_I2CEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)
Function Description	Configures I2C Digital noise filter.
Parameters	hi2c: : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral. DigitalFilter: : Coefficient of digital noise filter between 0x00 and 0x0F.
Return values	HAL: status

23.1.6 HAL_I2CEx_EnableWakeUp

Function Name	HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)
Function Description	Enables I2C wakeup from stop mode.
Parameters	hi2c: : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	HAL: status

23.1.7 HAL_I2CEx_DisableWakeUp

Function Name	HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)
Function Description	Disables I2C wakeup from stop mode.
Parameters	hi2c: : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	HAL: status

23.1.8 HAL_I2CEx_EnableFastModePlus

Function Name	void HAL_I2CEx_EnableFastModePlus (uint32_t ConfigFastModePlus)
Function Description	Enable the I2C fast mode plus driving capability.
Parameters	ConfigFastModePlus: selects the pin. This parameter can be one of the I2C Fast Mode Plus values
Return values	None:
Notes	- For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9. - For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C1 parameter. - For all I2C2 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C2 parameter.

- For all I2C3 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C3 parameter.

23.1.9 HAL_I2CEx_DisableFastModePlus

Function Name	void HAL_I2CEx_DisableFastModePlus (uint32_t ConfigFastModePlus)
Function Description	Disable the I2C fast mode plus driving capability.
Parameters	• ConfigFastModePlus: selects the pin. This parameter can be one of the I2C Fast Mode Plus values
Return values	• None:
Notes	• For I2C1, fast mode plus driving capability can be disabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9. • For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C1 parameter. • For all I2C2 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C2 parameter. • For all I2C3 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C3 parameter.

23.2 I2CEx Firmware driver defines

23.2.1 I2CEx

I2C Analog Filter Enabling

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2C Fast Mode Plus

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

I2C_FASTMODEPLUS_I2C3 Enable Fast Mode Plus on I2C3 pins

24 HAL I2S Generic Driver

24.1 I2S Firmware driver registers structures

24.1.1 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](#)

24.1.2 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_TypeDef* I2S_HandleTypeDef::Instance*
- *I2S_InitTypeDef I2S_HandleTypeDef::Init*
- *uint16_t* I2S_HandleTypeDef::pTxBuffPtr*
- *__IO uint16_t I2S_HandleTypeDef::TxXferSize*
- *__IO uint16_t I2S_HandleTypeDef::TxXferCount*
- *uint16_t* I2S_HandleTypeDef::pRxBuffPtr*
- *__IO uint16_t I2S_HandleTypeDef::RxXferSize*
- *__IO uint16_t I2S_HandleTypeDef::RxXferCount*
- *DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx*
- *DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx*
- *__IO HAL_LockTypeDef I2S_HandleTypeDef::Lock*
- *__IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State*
- *__IO uint32_t I2S_HandleTypeDef::ErrorCode*

24.2 I2S Firmware driver API description

24.2.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a I2S_HandleTypeDef handle structure.
2. Initialize the I2S low level resources by implement the HAL_I2S_MspInit() API:
 - a. Enable the SPIx interface clock.
 - b. I2S pins configuration:
 - Enable the clock for the I2S GPIOs.
 - Configure these I2S pins as alternate function.
 - c. NVIC configuration if you need to use interrupt process (HAL_I2S_Transmit_IT() and HAL_I2S_Receive_IT() APIs).
 - Configure the I2Sx interrupt priority.
 - Enable the NVIC I2S IRQ handle.
 - d. DMA Configuration if you need to use DMA process (HAL_I2S_Transmit_DMA() and HAL_I2S_Receive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx Channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx Channel.
 - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Channel.
3. Program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL_I2S_Init() function. The specific I2S interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_I2S_ENABLE_IT() and __HAL_I2S_DISABLE_IT() inside the transmit

and receive process. Make sure that either: External clock source is configured after setting correctly the define constant HSE_VALUE in the stm32l0xx_hal_conf.h file.

4. Three mode of operations are available within this driver :

Polling mode IO operation

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

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_I2S_Transmit_IT()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_I2S_Receive_IT()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_I2S_Transmit_DMA()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_I2S_Receive_DMA()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback
- Pause the DMA Transfer using HAL_I2S_DMAPause()
- Resume the DMA Transfer using HAL_I2S_DMAResume()
- Stop the DMA Transfer using HAL_I2S_DMAStop()

I2S HAL driver macros list

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

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



You can refer to the I2S HAL driver header file for more useful macros

24.2.2 Initialization and de-initialization functions

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

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

This section contains the following APIs:

- [*HAL_I2S_Init\(\)*](#)
- [*HAL_I2S_DeInit\(\)*](#)
- [*HAL_I2S_MspInit\(\)*](#)
- [*HAL_I2S_MspDeInit\(\)*](#)

24.2.3 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\(\)](#)

24.2.4 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\(\)](#)

24.2.5 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	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• HAL: status

24.2.6 HAL_I2S_DeInit

Function Name	HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)
Function Description	DeInitializes the I2S peripheral.
Parameters	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• HAL: status

24.2.7 HAL_I2S_MspInit

Function Name	void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)
Function Description	I2S MSP Init.
Parameters	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• None:

24.2.8 HAL_I2S_MspDeInit

Function Name	void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)
Function Description	I2S MSP DeInit.
Parameters	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• None:

24.2.9 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	• 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	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming). • This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

24.2.10 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	• 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	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming). • In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continouse way and as the I2S is not disabled at the end of the I2S transaction. • This function can use an Audio Frequency up to 44KHz when I2S Clock Source is 32MHz

24.2.11 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	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to data buffer. • Size: number of data sample to be sent:
Return values	• HAL: status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming). • This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

24.2.12 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	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Receive data buffer. • Size: number of data sample to be sent:
Return values	• HAL: status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size

parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.

- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- It is recommended to use DMA for the I2S receiver to avoid de-synchronisation between Master and Slave otherwise the I2S interrupt should be optimized.
- This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

24.2.13 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	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Transmit data buffer. • Size: number of data sample to be sent:
Return values	• HAL: status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

24.2.14 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	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Receive data buffer. • Size: number of data sample to be sent:
Return values	• HAL: status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the

clock de-synchronization between Master and Slave(example: audio streaming).

24.2.15 HAL_I2S_DMAPause

Function Name	HAL_StatusTypeDef HAL_I2S_DMAPause (I2S_HandleTypeDef * hi2s)
Function Description	Pauses the audio stream playing from the Media.
Parameters	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• HAL: status

24.2.16 HAL_I2S_DMAResume

Function Name	HAL_StatusTypeDef HAL_I2S_DMAResume (I2S_HandleTypeDef * hi2s)
Function Description	Resumes the audio stream playing from the Media.
Parameters	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• HAL: status

24.2.17 HAL_I2S_DMAStop

Function Name	HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)
Function Description	Stops the audio stream playing from the Media.
Parameters	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• HAL: status

24.2.18 HAL_I2S_IRQHandler

Function Name	void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)
Function Description	This function handles I2S interrupt request.
Parameters	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• None:

24.2.19 HAL_I2S_TxHalfCpltCallback

Function Name	void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Tx Transfer Half completed callbacks.
Parameters	• hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• None:

24.2.20 HAL_I2S_TxCpltCallback

Function Name	void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Tx Transfer completed callbacks.
Parameters	hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	None:

24.2.21 HAL_I2S_RxHalfCpltCallback

Function Name	void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Rx Transfer half completed callbacks.
Parameters	hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	None:

24.2.22 HAL_I2S_RxCpltCallback

Function Name	void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Rx Transfer completed callbacks.
Parameters	hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	None:

24.2.23 HAL_I2S_ErrorCallback

Function Name	void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)
Function Description	I2S error callbacks.
Parameters	hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	None:

24.2.24 HAL_I2S_GetState

Function Name	HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * hi2s)
Function Description	Return the I2S state.
Parameters	hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	HAL: state

24.2.25 HAL_I2S_GetError

Function Name	uint32_t HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)
Function Description	Return the I2S error code.

Parameters	hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	I2S: Error Code

24.3 I2S Firmware driver defines

24.3.1 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

I2S Clock Polarity

I2S_CPOL_LOW

I2S_CPOL_HIGH

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

I2S Error Code

HAL_I2S_ERROR_NONE No error

HAL_I2S_ERROR_UDR I2S Underrun error

HAL_I2S_ERROR_OVR I2S Overrun error

HAL_I2S_ERROR_FRE I2S Frame format error

HAL_I2S_ERROR_DMA DMA transfer error

I2S Exported Macros

`__HAL_I2S_RESET_HANDLE_STATE` **Description:**

- Reset I2S handle state.

Parameters:

- `__HANDLE__`: specifies the I2S Handle.

`__HAL_I2S_ENABLE`**Return value:**

- None

Description:

- Enable the specified SPI peripheral (in I2S mode).

Parameters:

- `__HANDLE__`: specifies the I2S Handle.

Return value:

- None

Description:

- Disable the specified SPI peripheral (in I2S mode).

Parameters:

- `__HANDLE__`: specifies the I2S Handle.

Return value:

- None

Description:

- Enable the specified I2S interrupts.

Parameters:

- `__HANDLE__`: specifies the I2S Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - `I2S_IT_TXE`: Tx buffer empty interrupt enable
 - `I2S_IT_RXNE`: RX buffer not empty interrupt enable
 - `I2S_IT_ERR`: Error interrupt enable

Return value:

- None

Description:

- Disable the specified I2S interrupts.

Parameters:

- `__HANDLE__`: specifies the I2S Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - `I2S_IT_TXE`: Tx buffer empty interrupt enable
 - `I2S_IT_RXNE`: RX buffer not empty interrupt enable
 - `I2S_IT_ERR`: Error interrupt enable

`__HAL_I2S_DISABLE``__HAL_I2S_ENABLE_IT``__HAL_I2S_DISABLE_IT`

__HAL_I2S_GET_IT_SOURCE**Return value:**

- None

Description:

- Checks if the specified I2S interrupt source is enabled or disabled.

Parameters:

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

Return value:

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

Description:

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

Parameters:

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

Return value:

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

Description:

- Clears the I2S OVR pending flag.

Parameters:

- **__HANDLE__**: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_GET_FLAG**__HAL_I2S_CLEAR_OVRFLAG**

`__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 Legacy***`I2S_STANDARD_PHILLIPS`***I2S MCLK Output***`I2S_MCLKOUTPUT_ENABLE``I2S_MCLKOUTPUT_DISABLE`***I2S Mode***`I2S_MODE_SLAVE_TX``I2S_MODE_SLAVE_RX``I2S_MODE_MASTER_TX``I2S_MODE_MASTER_RX`***I2S Standard***`I2S_STANDARD_PHILIPS``I2S_STANDARD_MSB``I2S_STANDARD_LSB``I2S_STANDARD_PCM_SHORT``I2S_STANDARD_PCM_LONG`

25 HAL IRDA Generic Driver

25.1 IRDA Firmware driver registers structures

25.1.1 IRDA_InitTypeDef

Data Fields

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

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate***
This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hirda->Init.BaudRate)))
- ***uint32_t IRDA_InitTypeDef::WordLength***
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [IRDAEx_Word_Length](#)
- ***uint32_t IRDA_InitTypeDef::Parity***
Specifies the parity mode. This parameter can be a value of [IRDA_Parity](#)
Note:When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint16_t IRDA_InitTypeDef::Mode***
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [IRDA_Transfer_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](#)

25.1.2 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`
- `IRDA_InitTypeDef IRDA_HandleTypeDef::Init`
- `uint8_t* IRDA_HandleTypeDef::pTxBuffPtr`
- `uint16_t IRDA_HandleTypeDef::TxXferSize`
- `uint16_t IRDA_HandleTypeDef::TxXferCount`
- `uint8_t* IRDA_HandleTypeDef::pRxBuffPtr`
- `uint16_t IRDA_HandleTypeDef::RxXferSize`
- `uint16_t IRDA_HandleTypeDef::RxXferCount`
- `uint16_t IRDA_HandleTypeDef::Mask`
- `DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmatx`
- `DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmarx`
- `HAL_LockTypeDef IRDA_HandleTypeDef::Lock`
- `__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::State`
- `__IO uint32_t IRDA_HandleTypeDef::ErrorCode`

25.2 IRDA Firmware driver API description

25.2.1 Initialization and Configuration functions

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

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit), the possible IRDA frame formats are as listed in the table below.
 - Power mode
 - Prescaler setting
 - Receiver/transmitter modes

Table 16: IRDA frame formats

M1 bit	M0 bit	PCE bit	IRDA frame
0	0	0	SB 8-bit data STB
0	0	1	SB 7-bit data PB STB
0	1	0	SB 9-bit data STB

M1 bit	M0 bit	PCE bit	IRDA frame
0	1	1	SB 8-bit data PB STB
1	0	0	SB 7-bit data STB
1	0	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_DeInit\(\)](#)
- [HAL_IRDA_MspInit\(\)](#)
- [HAL_IRDA_MspDeInit\(\)](#)

25.2.2 IO operation functions

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

- 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.
- Blocking mode API's are :
 - HAL_IRDA_Transmit()
 - HAL_IRDA_Receive()
- Non-Blocking mode API's with Interrupt are :
 - HAL_IRDA_Transmit_IT()
 - HAL_IRDA_Receive_IT()
 - HAL_IRDA_IRQHandler()
 - IRDA_Transmit_IT()
 - IRDA_Receive_IT()
- Non-Blocking mode functions with DMA are :
 - HAL_IRDA_Transmit_DMA()
 - HAL_IRDA_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in No_Blocking mode:
 - HAL_IRDA_TxCpltCallback()
 - 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_TxHalfCpltCallback\(\)](#)
- [HAL_IRDA_TxCpltCallback\(\)](#)
- [HAL_IRDA_RxHalfCpltCallback\(\)](#)
- [HAL_IRDA_RxCpltCallback\(\)](#)
- [HAL_IRDA_ErrorCallback\(\)](#)

25.2.3 Peripheral Control functions

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

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

This section contains the following APIs:

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

25.2.4 HAL_IRDA_Init

Function Name	HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)
Function Description	Initializes the IRDA mode according to the specified parameters in the <code>IRDA_InitTypeDef</code> and creates the associated handle .
Parameters	• hirda: IRDA handle
Return values	• HAL: status

25.2.5 HAL_IRDA_DeInit

Function Name	HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)
Function Description	DeInitializes the IRDA peripheral.
Parameters	• hirda: IRDA handle
Return values	• HAL: status

25.2.6 HAL_IRDA_MspInit

Function Name	void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)
Function Description	IRDA MSP Init.
Parameters	• hirda: IRDA handle
Return values	• None:

25.2.7 HAL_IRDA_MspDeInit

Function Name	void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)

Function Description	IRDA MSP Delnit.
Parameters	• hirda: IRDA handle
Return values	• None:

25.2.8 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	• hirda: IRDA handle • pData: pointer to data buffer • Size: amount of data to be sent • Timeout: Duration of the timeout
Return values	• HAL: status

25.2.9 HAL_IRDA_Receive

Function Name	HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	• hirda: IRDA handle • pData: pointer to data buffer • Size: amount of data to be received • Timeout: Duration of the timeout
Return values	• HAL: status

25.2.10 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	• hirda: IRDA handle • pData: pointer to data buffer • Size: amount of data to be sent
Return values	• HAL: status

25.2.11 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	• hirda: IRDA handle • pData: pointer to data buffer • Size: amount of data to be received

Return values

- **HAL:** status

25.2.12 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

- **hirda:** IRDA handle
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

25.2.13 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

- **hirda:** IRDA handle
- **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 contain the parity bit (MSB position)

25.2.14 HAL_IRDA_DMAPause

Function Name **HAL_StatusTypeDef HAL_IRDA_DMAPause**
(IRDA_HandleTypeDef * hirda)

Function Description Pauses the DMA Transfer.

Parameters

- **hirda:** pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

25.2.15 HAL_IRDA_DMAResume

Function Name **HAL_StatusTypeDef HAL_IRDA_DMAResume**
(IRDA_HandleTypeDef * hirda)

Function Description Resumes the DMA Transfer.

Parameters

- **hirda:** pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

25.2.16 HAL_IRDA_DMAStop

Function Name **HAL_StatusTypeDef HAL_IRDA_DMAStop**

(IRDA_HandleTypeDef * hirda)

Function Description	Stops the DMA Transfer.
Parameters	hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	HAL: status

25.2.17 HAL_IRDA_IRQHandler

Function Name	void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)
Function Description	This function handles IRDA interrupt request.
Parameters	hirda: IRDA handle
Return values	None:

25.2.18 HAL_IRDA_TxHalfCpltCallback

Function Name	void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)
Function Description	Tx Half Transfer completed callback.
Parameters	hirda: irda handle
Return values	None:

25.2.19 HAL_IRDA_TxCpltCallback

Function Name	void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)
Function Description	Tx Transfer completed callback.
Parameters	hirda: irda handle
Return values	None:

25.2.20 HAL_IRDA_RxHalfCpltCallback

Function Name	void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)
Function Description	Rx Half Transfer completed callback.
Parameters	hirda: irda handle
Return values	None:

25.2.21 HAL_IRDA_RxCpltCallback

Function Name	void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)
Function Description	Rx Transfer completed callback.
Parameters	hirda: irda handle

Return values

- **None:**

25.2.22 HAL_IRDA_ErrorCallback

Function Name **void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)**

Function Description IRDA error callback.

Parameters

- **hirda:** IRDA handle

Return values

- **None:**

25.2.23 HAL_IRDA_GetState

Function Name **HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)**

Function Description return the IRDA state

Parameters

- **hirda:** irda handle

Return values

- **HAL:** state

25.2.24 HAL_IRDA_GetError

Function Name **uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)**

Function Description Return the IRDA error code.

Parameters

- **hirda:** : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA.

Return values

- **IRDA:** Error Code

25.3 IRDA Firmware driver defines

25.3.1 IRDA

IRDA DMA RX

IRDA_DMA_RX_DISABLE

IRDA_DMA_RX_ENABLE

IS_IRDA_DMA_RX

IRDA DMA TX

IRDA_DMA_TX_DISABLE

IRDA_DMA_TX_ENABLE

IS_IRDA_DMA_TX

IRDA Exported Constants

HAL_IRDA_ERROR_NONE No error

HAL_IRDA_ERROR_PE Parity error

HAL_IRDA_ERROR_NE Noise error

HAL_IRDA_ERROR_FE frame error

HAL_IRDA_ERROR_ORE Overrun error
HAL_IRDA_ERROR_DMA DMA transfer error

IRDA Exported Macros

__HAL_IRDA_RESET_HANDLE_STATE **Description:**

- Reset IRDA handle state.

Parameters:

- **__HANDLE__**: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.

Return value:

- None

__HAL_IRDA_FLUSH_DRREGISTER **Description:**

- Flushes the IRDA DR register.

Parameters:

- **__HANDLE__**: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.

Return value:

- None

__HAL_IRDA_CLEAR_FLAG **Description:**

- Clears 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:

`__HAL_IRDA_CLEAR_FEFLAG`

- None

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 which can be USART1 or USART2. UART peripheral
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `IRDA_FLAG_REACK`: Receive

- enable acknowledge flag
- IRDA_FLAG_TEACK: Transmit enable acknowledge flag
- IRDA_FLAG_BUSY: Busy flag
- IRDA_FLAG_ABRF: Auto Baud rate detection flag
- IRDA_FLAG_ABRE: Auto Baud rate detection error flag
- IRDA_FLAG_TXE: Transmit data register empty flag
- IRDA_FLAG_TC: Transmission Complete flag
- IRDA_FLAG_RXNE: Receive data register not empty flag
- IRDA_FLAG_IDLE: Idle Line detection flag
- IRDA_FLAG_ORE: OverRun Error flag
- IRDA_FLAG_NE: Noise Error flag
- IRDA_FLAG_FE: Framing Error flag
- IRDA_FLAG_PE: Parity Error flag

Return value:

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

Description:

- Enable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2. 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

Description:

- Disable the specified IRDA interrupt.

`__HAL_IRDA_ENABLE_IT``__HAL_IRDA_DISABLE_IT`

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- `__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

Description:

- Check whether the specified IRDA interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- `__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).

Description:

- Check whether the specified IRDA

`__HAL_IRDA_GET_IT``__HAL_IRDA_GET_IT_SOURCE`

interrupt source is enabled.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- `__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).

Description:

- Clear the specified IRDA ISR flag, in setting the proper ICR register flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- `__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_CLEAR_IT`

__HAL_IRDA_SEND_REQ**Description:**

- Set a specific IRDA request flag.

Parameters:

- **__HANDLE__**: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- **__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_ENABLE**Description:**

- Enables the IRDA one bit sample method.

Parameters:

- **__HANDLE__**: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_DISABLE**Description:**

- Disables the IRDA one bit sample method.

Parameters:

- **__HANDLE__**: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ENABLE**Description:**

- Enable UART/USART associated to IRDA Handle.

Parameters:

- **__HANDLE__**: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.

Return value:

- None

__HAL_IRDA_DISABLE**Description:**

- Disable UART/USART associated to IRDA Handle.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.

Return value:

- None

Description:

- Ensure that IRDA Baud rate is less or equal to maximum value.

Parameters:

- `__BAUDRATE__`: specifies the IRDA Baudrate set by the user.

Return value:

- True: or False

Description:

- Ensure that IRDA prescaler value is strictly larger than 0.

Parameters:

- `__PRESCALER__`: specifies the IRDA prescaler value set by the user.

Return value:

- True: or False

IS_IRDA_BAUDRATE

IS_IRDA_PRESCALER

IRDA Flags

IRDA_FLAG_REACK	Receive Enable Acknowledge Flag
IRDA_FLAG_TEACK	Transmit Enable Acknowledge Flag
IRDA_FLAG_BUSY	Busy Flag
IRDA_FLAG_ABRF	Auto-Baud Rate Flag
IRDA_FLAG_ABRE	Auto-Baud Rate Error
IRDA_FLAG_TXE	Transmit Data Register Empty
IRDA_FLAG_TC	Transmission Complete
IRDA_FLAG_RXNE	Read Data Register Not Empty
IRDA_FLAG_ORE	OverRun Error
IRDA_FLAG_NE	Noise detected Flag
IRDA_FLAG_FE	Framing Error
IRDA_FLAG_PE	Parity Error

IRDA Interruption mask

IRDA_IT_MASK

IRDA Interrupt definition

IRDA_IT_PE

IRDA_IT_TXE

IRDA_IT_TC

IRDA_IT_RXNE

IRDA_IT_IDLE

IRDA_IT_ERR

IRDA_IT_ORE

IRDA_IT_NE

IRDA_IT_FE

IRDA Interrupt clear flag

IRDA_CLEAR_PEF Parity Error Clear Flag

IRDA_CLEAR_FEF Framing Error Clear Flag

IRDA_CLEAR_NEF Noise detected Clear Flag

IRDA_CLEAR_OREF OverRun Error Clear Flag

IRDA_CLEAR_TCF Transmission Complete Clear Flag

IRDA_CLEAR_IDLEF IDLE line detected Clear Flag

IRDA low power

IRDA_POWERMODE_NORMAL

IRDA_POWERMODE_LOWPOWER

IS_IRDA_POWERMODE

IRDA Mode

IRDA_MODE_DISABLE

IRDA_MODE_ENABLE

IS_IRDA_MODE

IRDA One bit

IRDA_ONE_BIT_SAMPLE_DISABLE

IRDA_ONE_BIT_SAMPLE_ENABLE

IS_IRDA_ONE_BIT_SAMPLE

IRDA Parity

IRDA_PARITY_NONE

IRDA_PARITY_EVEN

IRDA_PARITY_ODD

IS_IRDA_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
IS_IRDA_REQUEST_PARAMETER	

IRDA State

IRDA_STATE_DISABLE
IRDA_STATE_ENABLE
IS_IRDA_STATE

IRDA transfer mode

IRDA_MODE_RX
IRDA_MODE_TX
IRDA_MODE_TX_RX
IS_IRDA_TX_RX_MODE

26 HAL IRDA Extension Driver

26.1 IRDAEx Firmware driver defines

26.1.1 IRDAEx

IRDAEx Exported Macros

`IRDA_GETCLOCKSOURCE`

Description:

- Reports the IRDA clock source.

Parameters:

- `__HANDLE__`: specifies the UART Handle
- `__CLOCKSOURCE__`: output variable

Return value:

- IRDA: clocking source, written in `__CLOCKSOURCE__`.

`IRDA_MASK_COMPUTATION`

Description:

- Reports the mask to apply to retrieve the received data according to the word length and to the parity bits activation.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle

Return value:

- mask: to apply to USART RDR register value.

IRDAEx Word length

`IRDA_WORDLENGTH_7B`

`IRDA_WORDLENGTH_8B`

`IRDA_WORDLENGTH_9B`

`IS_IRDA_WORD_LENGTH`

27 HAL IWDG Generic Driver

27.1 IWDG Firmware driver registers structures

27.1.1 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 = 0xFFFF
- ***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 = 0xFFFF

27.1.2 IWDG_HandleTypeDef

Data Fields

- *IWDG_TypeDef * Instance*
- *IWDG_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_IWDG_StateTypeDef State*

Field Documentation

- ***IWDG_TypeDef* IWDG_HandleTypeDef::Instance***
Register base address
- ***IWDG_InitTypeDef IWDG_HandleTypeDef::Init***
IWDG required parameters
- ***HAL_LockTypeDef IWDG_HandleTypeDef::Lock***
IWDG Locking object
- ***__IO HAL_IWDG_StateTypeDef IWDG_HandleTypeDef::State***
IWDG communication state

27.2 IWDG Firmware driver API description

27.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by its own dedicated Low-Speed clock (LSI) and thus stays active even if the main clock fails. Once the IWDG is started, the LSI is forced ON and cannot be disabled (LSI cannot be disabled too), and the counter starts counting down from the reset value of 0xFFFF. When it reaches the end of count value (0x000) a system reset is generated.
- The IWDG counter should be refreshed at regular intervals, otherwise the watchdog generates an MCU reset when the counter reaches 0.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake-up from STANDBY). IWDGRST flag in RCC_CSR register can be used to inform when an IWDG reset occurs.

Min-max timeout value @32KHz (LSI): ~0.512ms / ~32.0s The IWDG timeout may vary due to LSI frequency dispersion. STM32L0xx devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM5 CH4 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

27.2.2 How to use this driver

If Window option is disabled

- Use IWDG using HAL_IWDG_Init() function to :
 - Enable write access to IWDG_PR, IWDG_RLR.
 - Configure the IWDG prescaler, counter reload value. This reload value will be loaded in the IWDG counter each time the counter is reloaded, then the IWDG will start counting down from this value.
- Use IWDG using HAL_IWDG_Start() function to :
 - Reload IWDG counter with value defined in the IWDG_RLR register.
 - Start the IWDG, when the IWDG is used in software mode (no need to enable the LSI, it will be enabled by hardware).
- Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_IWDG_Refresh() function.

if Window option is enabled:

- Use IWDG using HAL_IWDG_Start() function to enable IWDG downcounter
- Use IWDG using HAL_IWDG_Init() function to :
 - Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.
 - Configure the IWDG prescaler, reload value and window value.
- Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_IWDG_Refresh() function.

IWDG HAL driver macros list

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

- `__HAL_IWDG_START`: Enable the IWDG peripheral
- `__HAL_IWDG_RELOAD_COUNTER`: Reloads IWDG counter with value defined in the reload register

- IWDG_ENABLE_WRITE_ACCESS : Enable write access to IWDG_PR and IWDG_RLR registers
- __HAL_IWDG_DISABLE_WRITE_ACCESS : Disable write access to IWDG_PR and IWDG_RLR registers
- __HAL_IWDG_GET_FLAG: Get the selected IWDG's flag status

27.2.3 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

This section contains the following APIs:

- [HAL_IWDG_Init\(\)](#)
- [HAL_IWDG_MspInit\(\)](#)

27.2.4 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\(\)](#)

27.2.5 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_IWDG_GetState\(\)](#)

27.2.6 HAL_IWDG_Init

Function Name	HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)
Function Description	Initializes the IWDG according to the specified parameters in the IWDG_InitTypeDef and creates the associated handle.
Parameters	• hiwdg: : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• HAL: status

27.2.7 HAL_IWDG_MspInit

Function Name	void HAL_IWDG_MspInit (IWDG_HandleTypeDef * hiwdg)
Function Description	Initializes the IWDG MSP.
Parameters	• hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG

module.

Return values

- **None:**

27.2.8 HAL_IWDG_Start

Function Name **HAL_StatusTypeDef HAL_IWDG_Start (IWDG_HandleTypeDef * hiwdg)**

Function Description Starts the IWDG.

Parameters

- **hiwdg:** : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

27.2.9 HAL_IWDG_Refresh

Function Name **HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)**

Function Description Refreshes the IWDG.

Parameters

- **hiwdg:** : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

27.2.10 HAL_IWDG_GetState

Function Name **HAL_IWDG_StateTypeDef HAL_IWDG_GetState (IWDG_HandleTypeDef * hiwdg)**

Function Description Returns the IWDG state.

Parameters

- **hiwdg:** : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** state

27.3 IWDG Firmware driver defines

27.3.1 IWDG

IWDG Disable

IWDG_WINDOW_DISABLE

IWDG Exported Constants

IS_IWDG_KR

IS_IWDG_PRESCALER

IS_IWDG_RELOAD

IS_IWDG_WINDOW

IWDG Exported Macros

<code>__HAL_IWDG_RESET_HANDLE_STATE</code>	Description: - Reset IWDG handle state. Parameters: <code>__HANDLE__</code>: : IWDG handle Return value: - None
<code>__HAL_IWDG_START</code>	Description: - Enables the IWDG peripheral. Parameters: <code>__HANDLE__</code>: : IWDG handle Return value: - None
<code>__HAL_IWDG_RELOAD_COUNTER</code>	Description: - Reloads IWDG counter with value defined in the reload register (write access to IWDG_PR and IWDG_RLR registers disabled). Parameters: <code>__HANDLE__</code>: : IWDG handle Return value: - None
<code>IWDG_ENABLE_WRITE_ACCESS</code>	Description: - Enables write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers. Parameters: <code>__HANDLE__</code>: : IWDG handle Return value: - None
<code>IWDG_DISABLE_WRITE_ACCESS</code>	Description: - Disables write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers. Parameters: <code>__HANDLE__</code>: : IWDG handle Return value: - None
<code>__HAL_IWDG_GET_FLAG</code>	Description: Gets the selected IWDG's flag status.

Parameters:

- `__HANDLE__` : IWDG handle
- `__FLAG__` : specifies the flag to check.
This parameter can be one of the following values:
 - `IWDG_FLAG_PVU`: Watchdog counter reload value update flag
 - `IWDG_FLAG_RVU`: Watchdog counter prescaler value flag
 - `IWDG_FLAG_WVU`: Watchdog counter window value flag

Return value:

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

IWDG Flag definition

<code>IWDG_FLAG_PVU</code>	Watchdog counter prescaler value update flag
<code>IWDG_FLAG_RVU</code>	Watchdog counter reload value update flag
<code>IWDG_FLAG_WVU</code>	Watchdog counter window value update Flag

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
<code>IWDG_PRESCALER_256</code>	IWDG prescaler set to 256

IWDG key

<code>IWDG_KEY_RELOAD</code>	IWDG Reload Counter Enable
<code>IWDG_KEY_ENABLE</code>	IWDG Peripheral Enable
<code>IWDG_KEY_WRITE_ACCESS_ENABLE</code>	IWDG KR Write Access Enable
<code>IWDG_KEY_WRITE_ACCESS_DISABLE</code>	IWDG KR Write Access Disable

28 HAL LCD Generic Driver

28.1 LCD Firmware driver registers structures

28.1.1 LCD_InitTypeDef

Data Fields

- *uint32_t Prescaler*
- *uint32_t Divider*
- *uint32_t Duty*
- *uint32_t Bias*
- *uint32_t VoltageSource*
- *uint32_t Contrast*
- *uint32_t DeadTime*
- *uint32_t PulseOnDuration*
- *uint32_t HighDrive*
- *uint32_t BlinkMode*
- *uint32_t BlinkFrequency*
- *uint32_t MuxSegment*

Field Documentation

- ***uint32_t LCD_InitTypeDef::Prescaler***
Configures the LCD Prescaler. This parameter can be one value of [LCD_Prescaler](#)
- ***uint32_t LCD_InitTypeDef::Divider***
Configures the LCD Divider. This parameter can be one value of [LCD_Divider](#)
- ***uint32_t LCD_InitTypeDef::Duty***
Configures the LCD Duty. This parameter can be one value of [LCD_Duty](#)
- ***uint32_t LCD_InitTypeDef::Bias***
Configures the LCD Bias. This parameter can be one value of [LCD_Bias](#)
- ***uint32_t LCD_InitTypeDef::VoltageSource***
Selects the LCD Voltage source. This parameter can be one value of [LCD_Voltage_Source](#)
- ***uint32_t LCD_InitTypeDef::Contrast***
Configures the LCD Contrast. This parameter can be one value of [LCD_Contrast](#)
- ***uint32_t LCD_InitTypeDef::DeadTime***
Configures the LCD Dead Time. This parameter can be one value of [LCD_DeadTime](#)
- ***uint32_t LCD_InitTypeDef::PulseOnDuration***
Configures the LCD Pulse On Duration. This parameter can be one value of [LCD_PulseOnDuration](#)
- ***uint32_t LCD_InitTypeDef::HighDrive***
Configures the LCD High Drive. This parameter can be one value of [LCD_HighDrive](#)
- ***uint32_t LCD_InitTypeDef::BlinkMode***
Configures the LCD Blink Mode. This parameter can be one value of [LCD_BlinkMode](#)
- ***uint32_t LCD_InitTypeDef::BlinkFrequency***
Configures the LCD Blink frequency. This parameter can be one value of [LCD_BlinkFrequency](#)

- ***uint32_t LCD_InitTypeDef::MuxSegment***
Enable or disable mux segment. This parameter can be one value of [LCD_MuxSegment](#)

28.1.2 LCD_HandleTypeDef

Data Fields

- ***LCD_TypeDef * Instance***
- ***LCD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***LCD_TypeDef* LCD_HandleTypeDef::Instance***
- ***LCD_InitTypeDef LCD_HandleTypeDef::Init***
- ***HAL_LockTypeDef LCD_HandleTypeDef::Lock***
- ***__IO HAL_LCD_StateTypeDef LCD_HandleTypeDef::State***
- ***__IO uint32_t LCD_HandleTypeDef::ErrorCode***

28.2 LCD Firmware driver API description

28.2.1 How to use this driver

The LCD HAL driver can be used as follow:

1. Declare a LCD_HandleTypeDef handle structure.
2. Prepare the initialization of the LCD low level resources by implementing your HAL_LCD_MspInit() API:
 - a. Enable the LCDCLK (same as RTCCLK): to configure the RTCCLK/LCDCLK, use the RCC function HAL_RCCEx_PeriphCLKConfig, indicating here RCC_PERIPHCLK_LCD and the selected clock source (HSE, LSI or LSE)
 - b. The frequency generator allows you to achieve various LCD frame rates starting from an LCD input clock frequency (LCDCLK) which can vary from 32 kHz up to 1 MHz.
 - c. LCD pins configuration: - Enable the clock for the LCD GPIOs - Configure these LCD pins as alternate function no-pull.
 - d. Enable the LCD interface clock.
3. Set the Prescaler, Divider, Blink mode, Blink Frequency Duty, Bias, Voltage Source, Dead Time, Pulse On Duration and Contrast in the hlcd Init structure.
4. Initialize the LCD registers by calling the HAL_LCD_Init() API.
 - a. The HAL_LCD_Init() API configures the low level Hardware (GPIO, CLOCK, ...etc) by calling the user customized HAL_LCD_MspInit() API.
5. After calling the HAL_LCD_Init() the LCD RAM memory is cleared
6. Optionally you can update the LCD configuration using these macros:
 - a. LCD High Drive using the `__HAL_LCD_HIGHDRIVER_ENABLE()` and `__HAL_LCD_HIGHDRIVER_DISABLE()` macros
 - b. LCD Pulse ON Duration using the `__HAL_LCD_PULSEONDURATION_CONFIG()` macro

- c. LCD Dead Time using the `__HAL_LCD_DEADTIME_CONFIG()` macro
- d. The LCD Blink mode and frequency using the `__HAL_LCD_BLINK_CONFIG()` macro
- e. The LCD Contrast using the `__HAL_LCD_CONTRAST_CONFIG()` macro
- 7. Write to the LCD RAM memory using the `HAL_LCD_Write()` API, this API can be called several times to update the different LCD RAM registers before calling `HAL_LCD_UpdateDisplayRequest()` API.
- 8. The `HAL_LCD_Clear()` API can be used to clear the LCD RAM memory.
- 9. When the LCD RAM memory is updated, enable the update display request calling the `HAL_LCD_UpdateDisplayRequest()` API.

LCD and low power modes: The LCD remain active during STOP mode.

28.2.2 Initialization and Configuration functions

This section contains the following APIs:

- [`HAL\_LCD\_DeInit\(\)`](#)
- [`HAL\_LCD\_Init\(\)`](#)
- [`HAL\_LCD\_MspDeInit\(\)`](#)
- [`HAL\_LCD\_MspInit\(\)`](#)

28.2.3 IO operation functions

Using its double buffer memory the LCD controller ensures the coherency of the displayed information without having to use interrupts to control LCD_RAM modification. The application software can access the first buffer level (LCD_RAM) through the APB interface. Once it has modified the LCD_RAM using the `HAL_LCD_Write()` API, it sets the UDR flag in the LCD_SR register using the `HAL_LCD_UpdateDisplayRequest()` API. This UDR flag (update display request) requests the updated information to be moved into the second buffer level (LCD_DISPLAY). This operation is done synchronously with the frame (at the beginning of the next frame), until the update is completed, the LCD_RAM is write protected and the UDR flag stays high. Once the update is completed another flag (UDD - Update Display Done) is set and generates an interrupt if the UDDIE bit in the LCD_FCR register is set. The time it takes to update LCD_DISPLAY is, in the worst case, one odd and one even frame. The update will not occur (UDR = 1 and UDD = 0) until the display is enabled (LCDEN = 1).

This section contains the following APIs:

- [`HAL\_LCD\_Write\(\)`](#)
- [`HAL\_LCD\_Clear\(\)`](#)
- [`HAL\_LCD\_UpdateDisplayRequest\(\)`](#)

28.2.4 Peripheral State functions

This subsection provides a set of functions allowing to control the LCD:

- `HAL_LCD_GetState()` API can be helpful to check in run-time the state of the LCD peripheral State.
- `HAL_LCD_GetError()` API to return the LCD error code.

This section contains the following APIs:

- [`HAL\_LCD\_GetState\(\)`](#)
- [`HAL\_LCD\_GetError\(\)`](#)

28.2.5 HAL_LCD_DeInit



Function Name	HAL_StatusTypeDef HAL_LCD_DeInit (LCD_HandleTypeDef * hlcd)
Function Description	DeInitializes the LCD peripheral.
Parameters	hlcd: LCD handle
Return values	HAL: status
28.2.6 HAL_LCD_Init	
Function Name	HAL_StatusTypeDef HAL_LCD_Init (LCD_HandleTypeDef * hlcd)
Function Description	Initializes the LCD peripheral according to the specified parameters in the LCD_InitStruct.
Parameters	hlcd: LCD handle
Return values	None:
Notes	This function can be used only when the LCD is disabled. The LCD HighDrive can be enabled/disabled using related macros up to user.
28.2.7 HAL_LCD_MspDeInit	
Function Name	void HAL_LCD_MspDeInit (LCD_HandleTypeDef * hlcd)
Function Description	LCD MSP DeInit.
Parameters	hlcd: LCD handle
Return values	None:
28.2.8 HAL_LCD_MspInit	
Function Name	void HAL_LCD_MspInit (LCD_HandleTypeDef * hlcd)
Function Description	LCD MSP Init.
Parameters	hlcd: LCD handle
Return values	None:
28.2.9 HAL_LCD_Write	
Function Name	HAL_StatusTypeDef HAL_LCD_Write (LCD_HandleTypeDef * hlcd, uint32_t RAMRegisterIndex, uint32_t RAMRegisterMask, uint32_t Data)
Function Description	Writes a word in the specific LCD RAM.
Parameters	hlcd: LCD handle RAMRegisterIndex: specifies the LCD RAM Register. This parameter can be one of the following values: LCD_RAM_REGISTER0: LCD RAM Register 0 LCD_RAM_REGISTER1: LCD RAM Register 1 LCD_RAM_REGISTER2: LCD RAM Register 2 LCD_RAM_REGISTER3: LCD RAM Register 3 LCD_RAM_REGISTER4: LCD RAM Register 4 LCD_RAM_REGISTER5: LCD RAM Register 5

5LCD_RAM_REGISTER6: LCD RAM Register
 6LCD_RAM_REGISTER7: LCD RAM Register
 7LCD_RAM_REGISTER8: LCD RAM Register
 8LCD_RAM_REGISTER9: LCD RAM Register
 9LCD_RAM_REGISTER10: LCD RAM Register
 10LCD_RAM_REGISTER11: LCD RAM Register
 11LCD_RAM_REGISTER12: LCD RAM Register
 12LCD_RAM_REGISTER13: LCD RAM Register
 13LCD_RAM_REGISTER14: LCD RAM Register
 14LCD_RAM_REGISTER15: LCD RAM Register 15

- **RAMRegisterMask:** specifies the LCD RAM Register Data Mask.
- **Data:** specifies LCD Data Value to be written.

Return values

- **None:**

Notes

- For LCD glass COM*SEG as 8*40 for example, the LCD common terminals COM[0,7] are mapped on 32bits LCD_RAM_REGISTER[0,14] according to rules: COM(n) spread on LCD_RAM_REGISTER(2*n) and LCD_RAM_REGISTER(2*n+1). The segment terminals SEG[0,39] of COM(n) correspond to LSB bits of related LCD_RAM_REGISTER(2*n)[0,31] and LCD_RAM_REGISTER(2*n+1)[0,7]

28.2.10 HAL_LCD_Clear

Function Name **HAL_StatusTypeDef HAL_LCD_Clear (LCD_HandleTypeDef * hlcd)**

Function Description Clears the LCD RAM registers.

Parameters

- **hlcd:** LCD handle

Return values

- **None:**

28.2.11 HAL_LCD_UpdateDisplayRequest

Function Name **HAL_StatusTypeDef HAL_LCD_UpdateDisplayRequest (LCD_HandleTypeDef * hlcd)**

Function Description Enables the Update Display Request.

Parameters

- **hlcd:** LCD handle

Return values

- **None:**

Notes

- Each time software modifies the LCD_RAM it must set the UDR bit to transfer the updated data to the second level buffer. The UDR bit stays set until the end of the update and during this time the LCD_RAM is write protected.
- When the display is disabled, the update is performed for all LCD_DISPLAY locations. When the display is enabled, the update is performed only for locations for which commons are active (depending on DUTY). For example if DUTY = 1/2, only the LCD_DISPLAY of COM0 and COM1 will be updated.

28.2.12 HAL_LCD_GetState



Function Name	HAL_LCD_StateTypeDef HAL_LCD_GetState (LCD_HandleTypeDef * hlcd)
Function Description	Returns the LCD state.
Parameters	hlcd: LCD handle
Return values	HAL: state

28.2.13 HAL_LCD_GetError

Function Name	uint32_t HAL_LCD_GetError (LCD_HandleTypeDef * hlcd)
Function Description	Return the LCD error code.
Parameters	hlcd: LCD handle
Return values	LCD: Error Code

28.3 LCD Firmware driver defines

28.3.1 LCD

LCD Bias

LCD_BIAS_1_4 1/4 Bias

LCD_BIAS_1_2 1/2 Bias

LCD_BIAS_1_3 1/3 Bias

IS_LCD_BIAS

LCD Blink Frequency

LCD_BLINKFREQUENCY_DIV8 The Blink frequency = fLCD/8

LCD_BLINKFREQUENCY_DIV16 The Blink frequency = fLCD/16

LCD_BLINKFREQUENCY_DIV32 The Blink frequency = fLCD/32

LCD_BLINKFREQUENCY_DIV64 The Blink frequency = fLCD/64

LCD_BLINKFREQUENCY_DIV128 The Blink frequency = fLCD/128

LCD_BLINKFREQUENCY_DIV256 The Blink frequency = fLCD/256

LCD_BLINKFREQUENCY_DIV512 The Blink frequency = fLCD/512

LCD_BLINKFREQUENCY_DIV1024 The Blink frequency = fLCD/1024

IS_LCD_BLINK_FREQUENCY

LCD Blink Mode

LCD_BLINKMODE_OFF Blink disabled

LCD_BLINKMODE_SEG0_COM0 Blink enabled on SEG[0], COM[0] (1 pixel)

LCD_BLINKMODE_SEG0_ALLCOM Blink enabled on SEG[0], all COM (up to 8 pixels according to the programmed duty)

LCD_BLINKMODE_ALLSEG_ALLCOM Blink enabled on all SEG and all COM (all pixels)

IS_LCD_BLINK_MODE

LCD Contrast

LCD_CONTRASTLEVEL_0	Maximum Voltage = 2.60V
LCD_CONTRASTLEVEL_1	Maximum Voltage = 2.73V
LCD_CONTRASTLEVEL_2	Maximum Voltage = 2.86V
LCD_CONTRASTLEVEL_3	Maximum Voltage = 2.99V
LCD_CONTRASTLEVEL_4	Maximum Voltage = 3.12V
LCD_CONTRASTLEVEL_5	Maximum Voltage = 3.25V
LCD_CONTRASTLEVEL_6	Maximum Voltage = 3.38V
LCD_CONTRASTLEVEL_7	Maximum Voltage = 3.51V

IS_LCD_CONTRAST

LCD Dead Time

LCD_DEADTIME_0	No dead Time
LCD_DEADTIME_1	One Phase between different couple of Frame
LCD_DEADTIME_2	Two Phase between different couple of Frame
LCD_DEADTIME_3	Three Phase between different couple of Frame
LCD_DEADTIME_4	Four Phase between different couple of Frame
LCD_DEADTIME_5	Five Phase between different couple of Frame
LCD_DEADTIME_6	Six Phase between different couple of Frame
LCD_DEADTIME_7	Seven Phase between different couple of Frame

IS_LCD_DEAD_TIME

LCD Divider

LCD_DIVIDER_16	LCD frequency = CLKPS/16
LCD_DIVIDER_17	LCD frequency = CLKPS/17
LCD_DIVIDER_18	LCD frequency = CLKPS/18
LCD_DIVIDER_19	LCD frequency = CLKPS/19
LCD_DIVIDER_20	LCD frequency = CLKPS/20
LCD_DIVIDER_21	LCD frequency = CLKPS/21
LCD_DIVIDER_22	LCD frequency = CLKPS/22
LCD_DIVIDER_23	LCD frequency = CLKPS/23
LCD_DIVIDER_24	LCD frequency = CLKPS/24
LCD_DIVIDER_25	LCD frequency = CLKPS/25
LCD_DIVIDER_26	LCD frequency = CLKPS/26
LCD_DIVIDER_27	LCD frequency = CLKPS/27
LCD_DIVIDER_28	LCD frequency = CLKPS/28
LCD_DIVIDER_29	LCD frequency = CLKPS/29
LCD_DIVIDER_30	LCD frequency = CLKPS/30
LCD_DIVIDER_31	LCD frequency = CLKPS/31

IS_LCD_DIVIDER
LCD Duty

LCD_DUTY_STATIC Static duty

LCD_DUTY_1_2 1/2 duty

LCD_DUTY_1_3 1/3 duty

LCD_DUTY_1_4 1/4 duty

LCD_DUTY_1_8 1/8 duty

IS_LCD_DUTY

LCD Error Code

HAL_LCD_ERROR_NONE No error

HAL_LCD_ERROR_FCRSF Synchro flag timeout error

HAL_LCD_ERROR_UDR Update display request flag timeout error

HAL_LCD_ERROR_UDD Update display done flag timeout error

HAL_LCD_ERROR_ENS LCD enabled status flag timeout error

HAL_LCD_ERROR_RDY LCD Booster ready timeout error

LCD Exported Macros**__HAL_LCD_RESET_HANDLE_STATE Description:**

- Reset LCD handle state.

Parameters:

- **__HANDLE__**: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_ENABLE**Description:**

- macros to enables or disables the LCD

Parameters:

- **__HANDLE__**: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_DISABLE**__HAL_LCD_HIGHDRIVER_ENABLE****Description:**

- Macros to enable or disable the low resistance divider.

Parameters:

- **__HANDLE__**: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_HIGHDRIVER_DISABLE
__HAL_LCD_PULSEONDURATION_CONFIG

Notes:

- When this mode is enabled, the PulseOn Duration (PON) have to be programmed to 1/CK_PS (LCD_PULSEONDURATION_1).

Description:

- Macro to configure the LCD pulses on duration.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __DURATION__: specifies the LCD pulse on duration in terms of CK_PS (prescaled LCD clock period) pulses. This parameter can be one of the following values:
 - LCD_PULSEONDURATION_0: 0 pulse
 - LCD_PULSEONDURATION_1: Pulse ON duration = 1/CK_PS
 - LCD_PULSEONDURATION_2: Pulse ON duration = 2/CK_PS
 - LCD_PULSEONDURATION_3: Pulse ON duration = 3/CK_PS
 - LCD_PULSEONDURATION_4: Pulse ON duration = 4/CK_PS
 - LCD_PULSEONDURATION_5: Pulse ON duration = 5/CK_PS
 - LCD_PULSEONDURATION_6: Pulse ON duration = 6/CK_PS
 - LCD_PULSEONDURATION_7: Pulse ON duration = 7/CK_PS

Return value:

- None

__HAL_LCD_DEADTIME_CONFIG

Description:

- Macro to configure the LCD dead time.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __DEADTIME__: specifies the LCD dead time. This parameter can be one of the following values:
 - LCD_DEADTIME_0: No dead Time
 - LCD_DEADTIME_1: One Phase between different couple of Frame
 - LCD_DEADTIME_2: Two Phase between different couple of Frame
 - LCD_DEADTIME_3: Three Phase between different couple of Frame
 - LCD_DEADTIME_4: Four Phase between different couple of Frame

- LCD_DEADTIME_5: Five Phase between different couple of Frame
- LCD_DEADTIME_6: Six Phase between different couple of Frame
- LCD_DEADTIME_7: Seven Phase between different couple of Frame

Return value:

- None

__HAL_LCD_CONTRAST_CONFIG**Description:**

- Macro to configure the LCD Contrast.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __CONTRAST__: specifies the LCD Contrast. This parameter can be one of the following values:
 - LCD_CONTRASTLEVEL_0: Maximum Voltage = 2.60V
 - LCD_CONTRASTLEVEL_1: Maximum Voltage = 2.73V
 - LCD_CONTRASTLEVEL_2: Maximum Voltage = 2.86V
 - LCD_CONTRASTLEVEL_3: Maximum Voltage = 2.99V
 - LCD_CONTRASTLEVEL_4: Maximum Voltage = 3.12V
 - LCD_CONTRASTLEVEL_5: Maximum Voltage = 3.25V
 - LCD_CONTRASTLEVEL_6: Maximum Voltage = 3.38V
 - LCD_CONTRASTLEVEL_7: Maximum Voltage = 3.51V

Return value:

- None

__HAL_LCD_BLINK_CONFIG**Description:**

- Macro to configure the LCD Blink mode and Blink frequency.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __BLINKMODE__: specifies the LCD blink mode. This parameter can be one of the following values:
 - LCD_BLINKMODE_OFF: Blink disabled
 - LCD_BLINKMODE_SEG0_COM0: Blink enabled on SEG[0], COM[0] (1 pixel)
 - LCD_BLINKMODE_SEG0_ALLCOM: Blink enabled on SEG[0], all COM (up

to 8 pixels according to the programmed duty)

- LCD_BLINKMODE_ALLSEG_ALLCOM: Blink enabled on all SEG and all COM (all pixels)
- **__BLINKFREQUENCY__**: specifies the LCD blink frequency.
 - LCD_BLINKFREQUENCY_DIV8: The Blink frequency = $f_{Lcd}/8$
 - LCD_BLINKFREQUENCY_DIV16: The Blink frequency = $f_{Lcd}/16$
 - LCD_BLINKFREQUENCY_DIV32: The Blink frequency = $f_{Lcd}/32$
 - LCD_BLINKFREQUENCY_DIV64: The Blink frequency = $f_{Lcd}/64$
 - LCD_BLINKFREQUENCY_DIV128: The Blink frequency = $f_{Lcd}/128$
 - LCD_BLINKFREQUENCY_DIV256: The Blink frequency = $f_{Lcd}/256$
 - LCD_BLINKFREQUENCY_DIV512: The Blink frequency = $f_{Lcd}/512$
 - LCD_BLINKFREQUENCY_DIV1024: The Blink frequency = $f_{Lcd}/1024$

Return value:

- None

Description:

- Enables or disables the specified LCD interrupt.

Parameters:

- **__HANDLE__**: specifies the LCD Handle.
- **__INTERRUPT__**: specifies the LCD interrupt source to be enabled or disabled. This parameter can be one of the following values:
 - LCD_IT_SOF: Start of Frame Interrupt
 - LCD_IT_UDD: Update Display Done Interrupt

Return value:

- None

Description:

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

Parameters:

- **__HANDLE__**: specifies the LCD Handle.
- **__IT__**: specifies the LCD interrupt source to check. This parameter can be one of the

`__HAL_LCD_ENABLE_IT`

`__HAL_LCD_DISABLE_IT`

`__HAL_LCD_GET_IT_SOURCE`

following values:

- LCD_IT_SOF: Start of Frame Interrupt
- LCD_IT_UDD: Update Display Done Interrupt.

Return value:

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

Notes:

- If the device is in STOP mode (PCLK not provided) UDD will not generate an interrupt even if UDDIE = 1. If the display is not enabled the UDD interrupt will never occur.

Description:

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

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __FLAG__: specifies the flag to check.
This parameter can be one of the following values:
 - LCD_FLAG_ENS: LCD Enabled flag.
It indicates the LCD controller status.

Return value:

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

Notes:

- The ENS bit is set immediately when the LCDEN bit in the LCD_CR goes from 0 to 1. On deactivation it reflects the real status of LCD so it becomes 0 at the end of the last displayed frame. LCD_FLAG_SOF: Start of Frame flag. This flag is set by hardware at the beginning of a new frame, at the same time as the display data is updated. LCD_FLAG_UDR: Update Display Request flag. LCD_FLAG_UDD: Update Display Done flag. LCD_FLAG_RDY: Step-up converter Ready flag. It indicates the status of the step-up converter. LCD_FLAG_FCRSF: LCD Frame Control Register Synchronization Flag. This flag is set by hardware each time the LCD_FCR register is updated in the LCDCLK domain.

Description:

- Clears the specified LCD pending flag.

`__HAL_LCD_GET_FLAG`

`__HAL_LCD_CLEAR_FLAG`

Parameters:

- `__HANDLE__`: specifies the LCD Handle.
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
 - `LCD_FLAG_SOF`: Start of Frame Interrupt
 - `LCD_FLAG_UDD`: Update Display Done Interrupt

Return value:

- None

LCD Flag

`LCD_FLAG_ENS`
`LCD_FLAG_SOF`
`LCD_FLAG_UDR`
`LCD_FLAG_UDD`
`LCD_FLAG_RDY`
`LCD_FLAG_FCRSF`

LCD HighDrive

`LCD_HIGHDRIVE_0` Low resistance Drive
`LCD_HIGHDRIVE_1` High resistance Drive
`IS_LCD_HIGHDRIVE`

LCD Interrupts

`LCD_IT_SOF`
`LCD_IT_UDD`

LCD Mux Segment

`LCD_MUXSEGMENT_DISABLE` SEG pin multiplexing disabled
`LCD_MUXSEGMENT_ENABLE` SEG[31:28] are multiplexed with SEG[43:40]
`IS_LCD_MUXSEGMENT`

LCD Prescaler

`LCD_PRESCALER_1` `CLKPS = LCDCLK`
`LCD_PRESCALER_2` `CLKPS = LCDCLK/2`
`LCD_PRESCALER_4` `CLKPS = LCDCLK/4`
`LCD_PRESCALER_8` `CLKPS = LCDCLK/8`
`LCD_PRESCALER_16` `CLKPS = LCDCLK/16`
`LCD_PRESCALER_32` `CLKPS = LCDCLK/32`
`LCD_PRESCALER_64` `CLKPS = LCDCLK/64`
`LCD_PRESCALER_128` `CLKPS = LCDCLK/128`
`LCD_PRESCALER_256` `CLKPS = LCDCLK/256`

LCD_PRESCALER_512	CLKPS = LCDCLK/512
LCD_PRESCALER_1024	CLKPS = LCDCLK/1024
LCD_PRESCALER_2048	CLKPS = LCDCLK/2048
LCD_PRESCALER_4096	CLKPS = LCDCLK/4096
LCD_PRESCALER_8192	CLKPS = LCDCLK/8192
LCD_PRESCALER_16384	CLKPS = LCDCLK/16384
LCD_PRESCALER_32768	CLKPS = LCDCLK/32768
IS_LCD_PRESCALER	

LCD Pulse On Duration

LCD_PULSEONDURATION_0	Pulse ON duration = 0 pulse
LCD_PULSEONDURATION_1	Pulse ON duration = 1/CK_PS
LCD_PULSEONDURATION_2	Pulse ON duration = 2/CK_PS
LCD_PULSEONDURATION_3	Pulse ON duration = 3/CK_PS
LCD_PULSEONDURATION_4	Pulse ON duration = 4/CK_PS
LCD_PULSEONDURATION_5	Pulse ON duration = 5/CK_PS
LCD_PULSEONDURATION_6	Pulse ON duration = 6/CK_PS
LCD_PULSEONDURATION_7	Pulse ON duration = 7/CK_PS
IS_LCD_PULSE_ON_DURATION	

LCD RAM Register

LCD_RAM_REGISTER0	LCD RAM Register 0
LCD_RAM_REGISTER1	LCD RAM Register 1
LCD_RAM_REGISTER2	LCD RAM Register 2
LCD_RAM_REGISTER3	LCD RAM Register 3
LCD_RAM_REGISTER4	LCD RAM Register 4
LCD_RAM_REGISTER5	LCD RAM Register 5
LCD_RAM_REGISTER6	LCD RAM Register 6
LCD_RAM_REGISTER7	LCD RAM Register 7
LCD_RAM_REGISTER8	LCD RAM Register 8
LCD_RAM_REGISTER9	LCD RAM Register 9
LCD_RAM_REGISTER10	LCD RAM Register 10
LCD_RAM_REGISTER11	LCD RAM Register 11
LCD_RAM_REGISTER12	LCD RAM Register 12
LCD_RAM_REGISTER13	LCD RAM Register 13
LCD_RAM_REGISTER14	LCD RAM Register 14
LCD_RAM_REGISTER15	LCD RAM Register 15
IS_LCD_RAM_REGISTER	

LCD Voltage Source

LCD_VOLTAGESOURCE_INTERNAL	Internal voltage source for the LCD
LCD_VOLTAGESOURCE_EXTERNAL	External voltage source for the LCD
IS_LCD_VOLTAGE_SOURCE	

29 HAL LPTIM Generic Driver

29.1 LPTIM Firmware driver registers structures

29.1.1 LPTIM_ClockConfigTypeDef

Data Fields

- *uint32_t Source*
- *uint32_t Prescaler*

Field Documentation

- *uint32_t LPTIM_ClockConfigTypeDef::Source*
Selects the clock source. This parameter can be a value of [LPTIM_Clock_Source](#)
- *uint32_t LPTIM_ClockConfigTypeDef::Prescaler*
Specifies the counter clock Prescaler. This parameter can be a value of [LPTIM_Clock_Prescaler](#)

29.1.2 LPTIM_ULPClockConfigTypeDef

Data Fields

- *uint32_t Polarity*
- *uint32_t SampleTime*

Field Documentation

- *uint32_t LPTIM_ULPClockConfigTypeDef::Polarity*
Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of [LPTIM_Clock_Polarity](#)
- *uint32_t LPTIM_ULPClockConfigTypeDef::SampleTime*
Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of [LPTIM_Clock_Sample_Time](#)

29.1.3 LPTIM_TriggerConfigTypeDef

Data Fields

- *uint32_t Source*
- *uint32_t ActiveEdge*
- *uint32_t SampleTime*

Field Documentation

- ***uint32_t LPTIM_TriggerConfigTypeDef::Source***
Selects the Trigger source. This parameter can be a value of [LPTIMEx_Trigger_Source](#)
- ***uint32_t LPTIM_TriggerConfigTypeDef::ActiveEdge***
Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of [LPTIM_External_Trigger_Polarity](#)
- ***uint32_t LPTIM_TriggerConfigTypeDef::SampleTime***
Selects the trigger sampling time to configure the clock glitch filter. Note: This parameter is used only when an external trigger is used. This parameter can be a value of [LPTIM_Trigger_Sample_Time](#)

29.1.4 LPTIM_InitTypeDef**Data Fields**

- ***LPTIM_ClockConfigTypeDef Clock***
- ***LPTIM_ULPClockConfigTypeDef UltraLowPowerClock***
- ***LPTIM_TriggerConfigTypeDef Trigger***
- ***uint32_t OutputPolarity***
- ***uint32_t UpdateMode***
- ***uint32_t CounterSource***

Field Documentation

- ***LPTIM_ClockConfigTypeDef LPTIM_InitTypeDef::Clock***
Specifies the clock parameters
- ***LPTIM_ULPClockConfigTypeDef LPTIM_InitTypeDef::UltraLowPowerClock***
Specifies the Ultra Low Power clock parameters
- ***LPTIM_TriggerConfigTypeDef LPTIM_InitTypeDef::Trigger***
Specifies the Trigger parameters
- ***uint32_t LPTIM_InitTypeDef::OutputPolarity***
Specifies the Output polarity. This parameter can be a value of [LPTIM_Output_Polarity](#)
- ***uint32_t LPTIM_InitTypeDef::UpdateMode***
Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of [LPTIM_Updating_Mode](#)
- ***uint32_t LPTIM_InitTypeDef::CounterSource***
Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of [LPTIM_Counter_Source](#)

29.1.5 LPTIM_HandleTypeDef**Data Fields**

- ***LPTIM_TypeDef * Instance***
- ***LPTIM_InitTypeDef Init***

- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LPTIM_StateTypeDef State***

Field Documentation

- ***LPTIM_TypeDef* LPTIM_HandleTypeDef::Instance***
Register base address
- ***LPTIM_InitTypeDef LPTIM_HandleTypeDef::Init***
LPTIM required parameters
- ***HAL_StatusTypeDef LPTIM_HandleTypeDef::Status***
LPTIM peripheral status
- ***HAL_LockTypeDef LPTIM_HandleTypeDef::Lock***
LPTIM locking object
- ***__IO HAL_LPTIM_StateTypeDef LPTIM_HandleTypeDef::State***
LPTIM peripheral state

29.2 LPTIM Firmware driver API description

29.2.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and creates the associated handle.
- DeInitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- DeInitialize LPTIM MSP.

This section contains the following APIs:

- [***HAL_LPTIM_Init\(\)***](#)
- [***HAL_LPTIM_DeInit\(\)***](#)
- [***HAL_LPTIM_MspInit\(\)***](#)
- [***HAL_LPTIM_MspDeInit\(\)***](#)

29.2.2 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.
- Stop the Set once mode.
- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.

This section contains the following APIs:

- [*HAL_LPTIM_PWM_Start\(\)*](#)
- [*HAL_LPTIM_PWM_Stop\(\)*](#)
- [*HAL_LPTIM_PWM_Start_IT\(\)*](#)
- [*HAL_LPTIM_PWM_Stop_IT\(\)*](#)
- [*HAL_LPTIM_OnePulse_Start\(\)*](#)
- [*HAL_LPTIM_OnePulse_Stop\(\)*](#)
- [*HAL_LPTIM_OnePulse_Start_IT\(\)*](#)
- [*HAL_LPTIM_OnePulse_Stop_IT\(\)*](#)
- [*HAL_LPTIM_SetOnce_Start\(\)*](#)
- [*HAL_LPTIM_SetOnce_Stop\(\)*](#)
- [*HAL_LPTIM_SetOnce_Start_IT\(\)*](#)
- [*HAL_LPTIM_SetOnce_Stop_IT\(\)*](#)
- [*HAL_LPTIM_Encoder_Start\(\)*](#)
- [*HAL_LPTIM_Encoder_Stop\(\)*](#)
- [*HAL_LPTIM_Encoder_Start_IT\(\)*](#)
- [*HAL_LPTIM_Encoder_Stop_IT\(\)*](#)
- [*HAL_LPTIM_TimeOut_Start\(\)*](#)
- [*HAL_LPTIM_TimeOut_Stop\(\)*](#)
- [*HAL_LPTIM_TimeOut_Start_IT\(\)*](#)
- [*HAL_LPTIM_TimeOut_Stop_IT\(\)*](#)
- [*HAL_LPTIM_Counter_Start\(\)*](#)
- [*HAL_LPTIM_Counter_Stop\(\)*](#)
- [*HAL_LPTIM_Counter_Start_IT\(\)*](#)
- [*HAL_LPTIM_Counter_Stop_IT\(\)*](#)

29.2.3 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare)value.

This section contains the following APIs:

- [*HAL_LPTIM_ReadCounter\(\)*](#)
- [*HAL_LPTIM_ReadAutoReload\(\)*](#)
- [*HAL_LPTIM_ReadCompare\(\)*](#)

29.2.4 LPTIM IRQ handler

This section provides LPTIM IRQ handler function.

This section contains the following APIs:

- [*HAL_LPTIM_IRQHandler\(\)*](#)
- [*HAL_LPTIM_CompareMatchCallback\(\)*](#)
- [*HAL_LPTIM_AutoReloadMatchCallback\(\)*](#)
- [*HAL_LPTIM_TriggerCallback\(\)*](#)
- [*HAL_LPTIM_CompareWriteCallback\(\)*](#)
- [*HAL_LPTIM_AutoReloadWriteCallback\(\)*](#)
- [*HAL_LPTIM_DirectionUpCallback\(\)*](#)
- [*HAL_LPTIM_DirectionDownCallback\(\)*](#)

29.2.5 Peripheral State functions

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

This section contains the following APIs:

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

29.2.6 HAL_LPTIM_Init

Function Name	HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hlptim)
Function Description	Initializes the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and creates the associated handle.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.7 HAL_LPTIM_DeInit

Function Name	HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hlptim)
Function Description	DeInitializes the LPTIM peripheral.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.8 HAL_LPTIM_MspInit

Function Name	void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)
Function Description	Initializes the LPTIM MSP.
Parameters	• hlptim: : LPTIM handle
Return values	• None:

29.2.9 HAL_LPTIM_MspDeInit

Function Name	void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)
Function Description	DeInitializes LPTIM MSP.
Parameters	• hlptim: : LPTIM handle
Return values	• None:

29.2.10 HAL_LPTIM_PWM_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM PWM generation.
Parameters	• hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse: : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• HAL: status

29.2.11 HAL_LPTIM_PWM_Stop

Function Name **HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim)**

Function Description Stops the LPTIM PWM generation.

Parameters

- **hlptim:** : LPTIM handle

Return values

- **HAL:** status

29.2.12 HAL_LPTIM_PWM_Start_IT

Function Name **HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**

Function Description Starts the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF
- **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF

Return values

- **HAL:** status

29.2.13 HAL_LPTIM_PWM_Stop_IT

Function Name **HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hlptim)**

Function Description Stops the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** : LPTIM handle

Return values

- **HAL:** status

29.2.14 HAL_LPTIM_OnePulse_Start

Function Name **HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)**

Function Description Starts the LPTIM One pulse generation.

Parameters

- **hlptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

29.2.15 HAL_LPTIM_OnePulse_Stop

Function Name **HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hlptim)**

Function Description Stops the LPTIM One pulse generation.

Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.16 HAL_LPTIM_OnePulse_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM One pulse generation in interrupt mode.
Parameters	• hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse: : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• HAL: status

29.2.17 HAL_LPTIM_OnePulse_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the LPTIM One pulse generation in interrupt mode.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.18 HAL_LPTIM_SetOnce_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM in Set once mode.
Parameters	• hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse: : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• HAL: status

29.2.19 HAL_LPTIM_SetOnce_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the LPTIM Set once mode.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.20 HAL_LPTIM_SetOnce_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM Set once mode in interrupt mode.
Parameters	• hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse: : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• HAL: status

29.2.21 HAL_LPTIM_SetOnce_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the LPTIM Set once mode in interrupt mode.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.22 HAL_LPTIM_Encoder_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)
Function Description	Starts the Encoder interface.
Parameters	• hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• HAL: status

29.2.23 HAL_LPTIM_Encoder_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the Encoder interface.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.24 HAL_LPTIM_Encoder_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)
Function Description	Starts the Encoder interface in interrupt mode.
Parameters	• hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• HAL: status

29.2.25 HAL_LPTIM_Encoder_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the Encoder interface in nterrupt mode.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.26 HAL_LPTIM_TimeOut_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)
Function Description	Starts the Timeout function.
Parameters	• hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Timeout: : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• HAL: status

29.2.27 HAL_LPTIM_TimeOut_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the Timeout function.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.28 HAL_LPTIM_TimeOut_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)
Function Description	Starts the Timeout function in interrupt mode.
Parameters	• hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Timeout: : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• HAL: status

29.2.29 HAL_LPTIM_TimeOut_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function Description	Stops the Timeout function in interrupt mode.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.30 HAL_LPTIM_Counter_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period)
Function Description	Starts the Counter mode.
Parameters	• hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• HAL: status

29.2.31 HAL_LPTIM_Counter_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the Counter mode.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.32 HAL_LPTIM_Counter_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period)
Function Description	Starts the Counter mode in interrupt mode.
Parameters	• hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	• HAL: status

29.2.33 HAL_LPTIM_Counter_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the Counter mode in interrupt mode.
Parameters	• hlptim: : LPTIM handle
Return values	• HAL: status

29.2.34 HAL_LPTIM_ReadCounter

Function Name	uint32_t HAL_LPTIM_ReadCounter (LPTIM_HandleTypeDef * hlptim)
Function Description	This function returns the current counter value.

Parameters	• hlptim: LPTIM handle
Return values	• Counter: value.

29.2.35 HAL_LPTIM_ReadAutoReload

Function Name	uint32_t HAL_LPTIM_ReadAutoReload (LPTIM_HandleTypeDef * hlptim)
Function Description	This function return the current Autoreload (Period) value.
Parameters	• hlptim: LPTIM handle
Return values	• Autoreload: value.

29.2.36 HAL_LPTIM_ReadCompare

Function Name	uint32_t HAL_LPTIM_ReadCompare (LPTIM_HandleTypeDef * hlptim)
Function Description	This function return the current Compare (Pulse) value.
Parameters	• hlptim: LPTIM handle
Return values	• Compare: value.

29.2.37 HAL_LPTIM_IRQHandler

Function Name	void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)
Function Description	This function handles LPTIM interrupt request.
Parameters	• hlptim: LPTIM handle
Return values	• None:

29.2.38 HAL_LPTIM_CompareMatchCallback

Function Name	void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)
Function Description	Compare match callback in non blocking mode.
Parameters	• hlptim: : LPTIM handle
Return values	• None:

29.2.39 HAL_LPTIM_AutoReloadMatchCallback

Function Name	void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hlptim)
Function Description	Autoreload match callback in non blocking mode.
Parameters	• hlptim: : LPTIM handle
Return values	• None:

29.2.40 HAL_LPTIM_TriggerCallback

Function Name	void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)
Function Description	Trigger detected callback in non blocking mode.
Parameters	• hlptim: : LPTIM handle
Return values	• None:

29.2.41 HAL_LPTIM_CompareWriteCallback

Function Name	void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hlptim)
Function Description	Compare write callback in non blocking mode.
Parameters	• hlptim: : LPTIM handle
Return values	• None:

29.2.42 HAL_LPTIM_AutoReloadWriteCallback

Function Name	void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hlptim)
Function Description	Autoreload write callback in non blocking mode.
Parameters	• hlptim: : LPTIM handle
Return values	• None:

29.2.43 HAL_LPTIM_DirectionUpCallback

Function Name	void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hlptim)
Function Description	Direction counter changed from Down to Up callback in non blocking mode.
Parameters	• hlptim: : LPTIM handle
Return values	• None:

29.2.44 HAL_LPTIM_DirectionDownCallback

Function Name	void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hlptim)
Function Description	Direction counter changed from Up to Down callback in non blocking mode.
Parameters	• hlptim: : LPTIM handle
Return values	• None:

29.2.45 HAL_LPTIM_GetState

Function Name	HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (LPTIM_HandleTypeDef * hlptim)
Function Description	Returns the LPTIM state.

Parameters	• hlptim: LPTIM handle
Return values	• HAL: state

29.3 LPTIM Firmware driver defines

29.3.1 LPTIM

Clock polarity

LPTIM_CLOCKPOLARITY_RISING

LPTIM_CLOCKPOLARITY_FALLING

LPTIM_CLOCKPOLARITY_RISING_FALLING

Prescaler

LPTIM_PRESCALER_DIV1

LPTIM_PRESCALER_DIV2

LPTIM_PRESCALER_DIV4

LPTIM_PRESCALER_DIV8

LPTIM_PRESCALER_DIV16

LPTIM_PRESCALER_DIV32

LPTIM_PRESCALER_DIV64

LPTIM_PRESCALER_DIV128

Clock sample time

LPTIM_CLOCKSAMPLETIME_DIRECTTRANSITION

LPTIM_CLOCKSAMPLETIME_2TRANSITIONS

LPTIM_CLOCKSAMPLETIME_4TRANSITIONS

LPTIM_CLOCKSAMPLETIME_8TRANSITIONS

Clock source

LPTIM_CLOCKSOURCE_APBCLK_LPOSC

LPTIM_CLOCKSOURCE_ULPTIM

Counter source

LPTIM_COUNTERSOURCE_INTERNAL

LPTIM_COUNTERSOURCE_EXTERNAL

LPTIM Exported constants

IS_LPTIM_AUTORELOAD

IS_LPTIM_COMPARE

IS_LPTIM_CLOCK_SOURCE

IS_LPTIM_CLOCK_PRESCALER

IS_LPTIM_CLOCK_PRESCALERDIV1

IS_LPTIM_OUTPUT_POLARITY

IS_LPTIM_CLOCK_SAMPLE_TIME
IS_LPTIM_CLOCK_POLARITY
IS_LPTIM_EXT_TRG_POLARITY
IS_LPTIM_TRIG_SAMPLE_TIME
IS_LPTIM_UPDATE_MODE
IS_LPTIM_COUNTER_SOURCE
IS_LPTIM_PERIOD
IS_LPTIM_PULSE

LPTIM Exported Macros

__HAL_LPTIM_RESET_HANDLE_STATE

Description:

- Reset LPTIM handle state.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_ENABLE

Description:

- Enable/Disable the LPTIM peripheral.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- None

__HAL_LPTIM_DISABLE

__HAL_LPTIM_START_CONTINUOUS

Description:

- Starts the LPTIM peripheral in Continuous or in single mode.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None

__HAL_LPTIM_START_SINGLE

__HAL_LPTIM_AUTORELOAD_SET

Description:

- Writes the passed parameter in the Autoreload register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Autoreload value

Return value:

__HAL_LPTIM_COMPARE_SET

- None

Description:

- Writes the passed parameter in the Compare register.

Parameters:

- **__HANDLE__**: LPTIM handle
- **__VALUE__**: Compare value

Return value:

- None

__HAL_LPTIM_GET_FLAG**Description:**

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

Parameters:

- **__HANDLE__**: LPTIM handle
- **__FLAG__**: LPTIM flag to check This parameter can be a value of:
 - LPTIM_FLAG_DOWN : Counter direction change up Flag.
 - LPTIM_FLAG_UP : Counter direction change down to up Flag.
 - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
 - LPTIM_FLAG_CMPOK : Compare register update OK Flag.
 - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
 - LPTIM_FLAG_ARRM : Autoreload match Flag.
 - LPTIM_FLAG_CMPM : Compare match Flag.

Return value:

- The: state of the specified flag (SET or RESET).

__HAL_LPTIM_CLEAR_FLAG**Description:**

- Clears the specified LPTIM flag.

Parameters:

- **__HANDLE__**: LPTIM handle.
- **__FLAG__**: LPTIM flag to clear. This parameter can be a value of:
 - LPTIM_FLAG_DOWN : Counter direction change up Flag.
 - LPTIM_FLAG_UP : Counter direction change down to up Flag.
 - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
 - LPTIM_FLAG_CMPOK : Compare

__HAL_LPTIM_ENABLE_IT

- register update OK Flag.
- LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
- LPTIM_FLAG_ARRM : Autoreload match Flag.
- LPTIM_FLAG_CMPM : Compare match Flag.

Return value:

- None.

Description:

- Enable the specified LPTIM interrupt.

Parameters:

- __HANDLE__ : LPTIM handle.
- __INTERRUPT__ : LPTIM interrupt to set. This parameter can be a value of:
 - LPTIM_IT_DOWN : Counter direction change up Interrupt.
 - LPTIM_IT_UP : Counter direction change down to up Interrupt.
 - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
 - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
 - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
 - LPTIM_IT_ARRM : Autoreload match Interrupt.
 - LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- None.

__HAL_LPTIM_DISABLE_IT**Description:**

- Disable the specified LPTIM interrupt.

Parameters:

- __HANDLE__ : LPTIM handle.
- __INTERRUPT__ : LPTIM interrupt to set. This parameter can be a value of:
 - LPTIM_IT_DOWN : Counter direction change up Interrupt.
 - LPTIM_IT_UP : Counter direction change down to up Interrupt.
 - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
 - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
 - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
 - LPTIM_IT_ARRM : Autoreload

match Interrupt.

- LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- None.

Description:

- Checks whether the specified LPTIM interrupt is set or not.

Parameters:

- __HANDLE__ : LPTIM handle.
- __INTERRUPT__ : LPTIM interrupt to check. This parameter can be a value of:
 - LPTIM_IT_DOWN : Counter direction change up Interrupt.
 - LPTIM_IT_UP : Counter direction change down to up Interrupt.
 - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
 - LPTIM_IT_CMPOK : Compare register update OK Interrupt.
 - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
 - LPTIM_IT_ARRM : Autoreload match Interrupt.
 - LPTIM_IT_CMPM : Compare match Interrupt.

Return value:

- Interrupt: status.

__HAL_LPTIM_GET_IT_SOURCE

Trigger polarity

LPTIM_ACTIVEEDGE_RISING

LPTIM_ACTIVEEDGE_FALLING

LPTIM_ACTIVEEDGE_RISING_FALLING

Flag definition

LPTIM_FLAG_DOWN

LPTIM_FLAG_UP

LPTIM_FLAG_ARROK

LPTIM_FLAG_CMPOK

LPTIM_FLAG_EXTTRIG

LPTIM_FLAG_ARRM

LPTIM_FLAG_CMPM

Interrupts definition

LPTIM_IT_DOWN

LPTIM_IT_UP
LPTIM_IT_ARROK
LPTIM_IT_CMPOK
LPTIM_IT_EXTTRIG
LPTIM_IT_ARRM
LPTIM_IT_CMPM

Output polarity

LPTIM_OUTPUTPOLARITY_HIGH
LPTIM_OUTPUTPOLARITY_LOW

Trigger sample time

LPTIM_TRIGSAMPLETIME_DIRECTTRANSITION
LPTIM_TRIGSAMPLETIME_2TRANSITIONS
LPTIM_TRIGSAMPLETIME_4TRANSITIONS
LPTIM_TRIGSAMPLETIME_8TRANSITIONS

Updating mode

LPTIM_UPDATE_IMMEDIATE
LPTIM_UPDATE_ENDOFPERIOD

30 HAL LPTIM Extension Driver

30.1 LPTIMEx Firmware driver defines

30.1.1 LPTIMEx

Trigger source

LPTIM_TRIGSOURCE_SOFTWARE

LPTIM_TRIGSOURCE_0

LPTIM_TRIGSOURCE_1

LPTIM_TRIGSOURCE_2

LPTIM_TRIGSOURCE_3

LPTIM_TRIGSOURCE_4

LPTIM_TRIGSOURCE_5

LPTIM_TRIGSOURCE_6

LPTIM_TRIGSOURCE_7

31 HAL PCD Generic Driver

31.1 PCD Firmware driver registers structures

31.1.1 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_Speed](#)
- *uint32_t PCD_InitTypeDef::ep0_mps*
Set the Endpoint 0 Max Packet size. This parameter can be any value of [PCD_USB_EP0_MPS](#)
- *uint32_t PCD_InitTypeDef::phy_iface*
Select the used PHY interface. This parameter can be any value of [PCD_USB_Core_PHY](#)
- *uint32_t PCD_InitTypeDef::Sof_enable*
Enable or disable the output of the SOF signal. This parameter can be set to ENABLE or DISABLE
- *uint32_t PCD_InitTypeDef::low_power_enable*
Enable or disable Low Power mode This parameter can be set to ENABLE or DISABLE
- *uint32_t PCD_InitTypeDef::lpm_enable*
Enable or disable Link Power Management. This parameter can be set to ENABLE or DISABLE
- *uint32_t PCD_InitTypeDef::battery_charging_enable*
Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE

31.1.2 PCD_EPTTypeDef

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_USB_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

31.1.3 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

31.2 PCD Firmware driver API description

31.2.1 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();***

31.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- [*HAL_PCD_Init\(\)*](#)
- [*HAL_PCD_DeInit\(\)*](#)
- [*HAL_PCD_MspInit\(\)*](#)
- [*HAL_PCD_MspDeInit\(\)*](#)

31.2.3 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\(\)*](#)

31.2.4 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\(\)*](#)
- [*PCD_WritePMA\(\)*](#)
- [*PCD_ReadPMA\(\)*](#)

31.2.5 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_PCD_GetState\(\)](#)

31.2.6 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	• hpcd: PCD handle
Return values	• HAL: status

31.2.7 HAL_PCD_DeInit

Function Name	HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)
Function Description	DeInitializes the PCD peripheral.
Parameters	• hpcd: PCD handle
Return values	• HAL: status

31.2.8 HAL_PCD_MspInit

Function Name	void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)
Function Description	Initializes the PCD MSP.
Parameters	• hpcd: PCD handle
Return values	• None:

31.2.9 HAL_PCD_MspDeInit

Function Name	void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)
Function Description	DeInitializes PCD MSP.
Parameters	• hpcd: PCD handle
Return values	• None:

31.2.10 HAL_PCD_Start

Function Name	HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)
Function Description	Start The USB OTG Device.
Parameters	• hpcd: PCD handle
Return values	• HAL: status

31.2.11 HAL_PCD_Stop

Function Name	HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)

Function Description Stop The USB OTG Device.

Parameters • **hpcd**: PCD handle

Return values • **HAL**: status

31.2.12 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

31.2.13 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**:

31.2.14 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**:

31.2.15 HAL_PCD_SetupStageCallback

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

Function Description Setup stage callback.

Parameters • **hpcd**: PCD handle

Return values • **None**:

31.2.16 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**:

31.2.17 HAL_PCD_ResetCallback

Function Name	void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)
Function Description	USB Reset callbacks.
Parameters	• hpcd: PCD handle
Return values	• None:

31.2.18 HAL_PCD_SuspendCallback

Function Name	void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)
Function Description	Suspend event callbacks.
Parameters	• hpcd: PCD handle
Return values	• None:

31.2.19 HAL_PCD_ResumeCallback

Function Name	void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)
Function Description	Resume event callbacks.
Parameters	• hpcd: PCD handle
Return values	• None:

31.2.20 HAL_PCD_ISOOUTIncompleteCallback

Function Name	void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)
Function Description	Incomplete ISO OUT callbacks.
Parameters	• hpcd: PCD handle • epnum: endpoint number
Return values	• None:

31.2.21 HAL_PCD_ISOINIncompleteCallback

Function Name	void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)
Function Description	Incomplete ISO IN callbacks.
Parameters	• hpcd: PCD handle • epnum: endpoint number
Return values	• None:

31.2.22 HAL_PCD_ConnectCallback

Function Name	void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)
Function Description	Connection event callbacks.
Parameters	• hpcd: PCD handle

Return values • **None:**

31.2.23 HAL_PCD_DisconnectCallback

Function Name void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)

Function Description Disconnection event callbacks.

Parameters • **hpcd:** PCD handle

Return values • **None:**

31.2.24 HAL_PCD_DevConnect

Function Name HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)

Function Description Connect the USB device.

Parameters • **hpcd:** PCD handle

Return values • **HAL:** status

31.2.25 HAL_PCD_DevDisconnect

Function Name HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)

Function Description Disconnect the USB device.

Parameters • **hpcd:** PCD handle

Return values • **HAL:** status

31.2.26 HAL_PCD_SetAddress

Function Name HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)

Function Description Set the USB Device address.

Parameters • **hpcd:** PCD handle
• **address:** new device address

Return values • **HAL:** status

31.2.27 HAL_PCD_EP_Open

Function Name HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)

Function Description Open and configure an endpoint.

Parameters • **hpcd:** PCD handle
• **ep_addr:** endpoint address
• **ep_mps:** endpoint max packert size
• **ep_type:** endpoint type

Return values • **HAL:** status

31.2.28 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	• hpcd: PCD handle • ep_addr: endpoint address
Return values	• HAL: status

31.2.29 HAL_PCD_EP_Receive

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)
Function Description	Receive an amount of data.
Parameters	• hpcd: PCD handle • ep_addr: endpoint address • pBuf: pointer to the reception buffer • len: amount of data to be received
Return values	• HAL: status

31.2.30 HAL_PCD_EP_GetRxCount

Function Name	uint16_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function Description	Get Received Data Size.
Parameters	• hpcd: PCD handle • ep_addr: endpoint address
Return values	• Data: Size

31.2.31 HAL_PCD_EP_Transmit

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)
Function Description	Send an amount of data.
Parameters	• hpcd: PCD handle • ep_addr: endpoint address • pBuf: pointer to the transmission buffer • len: amount of data to be sent
Return values	• HAL: status

31.2.32 HAL_PCD_EP_SetStall

Function Name	HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function Description	Set a STALL condition over an endpoint.

Parameters	• hpcd: PCD handle • ep_addr: endpoint address
Return values	• HAL: status

31.2.33 HAL_PCD_EP_ClrStall

Function Name	HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function Description	Clear a STALL condition over in an endpoint.
Parameters	• hpcd: PCD handle • ep_addr: endpoint address
Return values	• HAL: status

31.2.34 HAL_PCD_EP_Flush

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
Function Description	Flush an endpoint.
Parameters	• hpcd: PCD handle • ep_addr: endpoint address
Return values	• HAL: status

31.2.35 HAL_PCD_ActivateRemoteWakeup

Function Name	HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)
Function Description	HAL_PCD_ActivateRemoteWakeup : active remote wakeup signalling.
Parameters	• hpcd: PCD handle
Return values	• status:

31.2.36 HAL_PCD_DeActivateRemoteWakeup

Function Name	HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)
Function Description	HAL_PCD_DeActivateRemoteWakeup : de-active remote wakeup signalling.
Parameters	• hpcd: PCD handle
Return values	• status:

31.2.37 PCD_WritePMA

Function Name	void PCD_WritePMA (USB_TypeDef * USBx, uint8_t * pbUsrBuf, uint16_t wPMABufAddr, uint16_t wNBytes)
Function Description	Copy a buffer from user memory area to packet memory area (PMA)

Parameters	• USBx: USB device • pbUsrBuf: pointer to user memory area. • wPMABufAddr: address into PMA. • wNBytes: no. of bytes to be copied.
Return values	• None:

31.2.38 PCD_ReadPMA

Function Name	void PCD_ReadPMA (USB_TypeDef * USBx, uint8_t * pbUsrBuf, uint16_t wPMABufAddr, uint16_t wNBytes)
Function Description	Copy a buffer from user memory area to packet memory area (PMA)
Parameters	• USBx: USB device • pbUsrBuf: pointer to user memory area. • wPMABufAddr: address into PMA. • wNBytes: no. of bytes to be copied.
Return values	• None:

31.2.39 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

31.3 PCD Firmware driver defines

31.3.1 PCD

PCD End Point

PCD_ENDP0

PCD_ENDP1

PCD_ENDP2

PCD_ENDP3

PCD_ENDP4

PCD_ENDP5

PCD_ENDP6

PCD_ENDP7

PCD_SNG_BUF

PCD_DBL_BUF

IS_PCD_ALL_INSTANCE

PCD Interrupt

__HAL_PCD_GET_FLAG	
__HAL_PCD_CLEAR_FLAG	
USB_WAKEUP_EXTI_LINE	External interrupt line 18 Connected to the USB FS EXTI Line
__HAL_USB_WAKEUP_EXTI_ENABLE_IT	
__HAL_USB_WAKEUP_EXTI_DISABLE_IT	
__HAL_USB_EXTI_GENERATE_SWIT	
PCD_SET_ENDPOINT	
PCD_GET_ENDPOINT	
PCD_SET_EPTYPE	Description: - sets the type in the endpoint register(bits EP_TYPE[1:0]) Parameters: - USBx: USB device. - bEpNum: Endpoint Number. - wType: Endpoint Type. Return value: - None
PCD_GET_EPTYPE	Description: - gets the type in the endpoint register(bits EP_TYPE[1:0]) Parameters: - USBx: USB device. - bEpNum: Endpoint Number. Return value: - Endpoint: Type
PCD_FreeUserBuffer	Description: - free buffer used from the application realizing it to the line toggles bit SW_BUF in the double buffered endpoint register Parameters: - USBx: USB device. - bEpNum Return value: - None
PCD_GET_DB_DIR	Description: gets direction of the double buffered endpoint Parameters:

PCD_SET_EP_TX_STATUS

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- EP_DBUF_OUT: if the endpoint counter not yet programmed.

Description:

- sets the status for tx transfer (bits STAT_TX[1:0]).

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wState: new state

Return value:

- None

Description:

- sets the status for rx transfer (bits STAT_TX[1:0])

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wState: new state

Return value:

- None

Description:

- sets the status for rx & tx (bits STAT_TX[1:0] & STAT_RX[1:0])

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wStaterx: new state.
- wStatetx: new state.

Return value:

- None

Description:

- gets the status for tx/rx transfer (bits STAT_TX[1:0] /STAT_RX[1:0])

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:**PCD_SET_EP_RX_STATUS****PCD_SET_EP_TXRX_STATUS****PCD_GET_EP_TX_STATUS**

PCD_GET_EP_RX_STATUS

PCD_SET_EP_TX_VALID

- status

Description:

- sets directly the VALID tx/rx-status into the endpoint register

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_SET_EP_RX_VALID

PCD_GET_EP_TX_STALL_STATUS

Description:

- checks stall condition in an endpoint.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- TRUE: = endpoint in stall condition.

PCD_GET_EP_RX_STALL_STATUS

PCD_SET_EP_KIND

Description:

- set & clear EP_KIND bit.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_EP_KIND

PCD_SET_OUT_STATUS

Description:

- Sets/clears directly STATUS_OUT bit in the endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_OUT_STATUS

PCD_SET_EP_DBUF

Description:

- Sets/clears directly EP_KIND bit in the

endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_EP_DBUF
PCD_CLEAR_RX_EP_CTR

Description:

- Clears bit CTR_RX / CTR_TX in the endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_TX_EP_CTR
PCD_RX_DTOG

Description:

- Toggles DTOG_RX / DTOG_TX bit in the endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_TX_DTOG
PCD_CLEAR_RX_DTOG

Description:

- Clears DTOG_RX / DTOG_TX bit in the endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_TX_DTOG
PCD_SET_EP_ADDRESS

Description:

- Sets address in an endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

PCD_GET_EP_ADDRESS

- bAddr: Address.

Return value:

- None

Description:

- Gets address in an endpoint register.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_EP_TX_ADDRESS

PCD_EP_TX_CNT

PCD_EP_RX_ADDRESS

PCD_EP_RX_CNT

PCD_SET_EP_TX_ADDRESS

Description:

- sets address of the tx/rx buffer.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wAddr: address to be set (must be word aligned).

Return value:

- None

PCD_SET_EP_RX_ADDRESS

PCD_GET_EP_TX_ADDRESS

Description:

- Gets address of the tx/rx buffer.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- address: of the buffer.

PCD_GET_EP_RX_ADDRESS

PCD_CALC_BLK32

Description:

- Sets counter of rx buffer with no.

Parameters:

- dwReg: Register.
- wCount: Counter.
- wNBlocks: Nb of block

PCD_CALC_BLK2
PCD_SET_EP_CNT_RX_REG
PCD_SET_EP_RX_DBUF0_CNT
PCD_SET_EP_TX_CNT

Return value:

- None

Description:

- sets counter for the tx/rx buffer.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wCount: Counter value.

Return value:

- None

PCD_SET_EP_RX_CNT
PCD_GET_EP_TX_CNT

Description:

- gets counter of the tx buffer.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- Counter: value

PCD_GET_EP_RX_CNT
PCD_SET_EP_DBUF0_ADDR

Description:

- Sets buffer 0/1 address in a double buffer endpoint.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.

Return value:

- Counter: value

PCD_SET_EP_DBUF1_ADDR
PCD_SET_EP_DBUF_ADDR

Description:

- Sets addresses in a double buffer endpoint.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.

PCD_GET_EP_DBUF0_ADDR

- wBuf1Addr: = buffer 1 address.

Return value:

- None

Description:

- Gets buffer 0/1 address of a double buffer endpoint.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_GET_EP_DBUF1_ADDR

PCD_SET_EP_DBUF0_CNT

Description:

- Gets buffer 0/1 address of a double buffer endpoint.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.
- bDir: endpoint dir EP_DBUF_OUT = OUT and EP_DBUF_IN = IN
- wCount: Counter value

Return value:

- None

PCD_SET_EP_DBUF1_CNT

PCD_SET_EP_DBUF_CNT

PCD_GET_EP_DBUF0_CNT

Description:

- Gets buffer 0/1 rx/tx counter for double buffering.

Parameters:

- USBx: USB device.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_GET_EP_DBUF1_CNT

PCD Speed

PCD_SPEED_HIGH

PCD_SPEED_FULL

PCD USB Core PHY

PCD_PHY_EMBEDDED

PCD USB 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 USB EP Type

PCD_EP_TYPE_CTRL

PCD_EP_TYPE_ISOC

PCD_EP_TYPE_BULK

PCD_EP_TYPE_INTR

32 HAL PCD Extension Driver

32.1 PCDEx Firmware driver API description

32.1.1 Peripheral extended features functions

This section contains the following APIs:

- [HAL_PCDEx_PMAConfig\(\)](#)

32.1.2 HAL_PCDEx_PMAConfig

Function Name	HAL_StatusTypeDef HAL_PCDEx_PMAConfig (PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t ep_kind, uint32_t pmaaddress)
Function Description	Configure PMA for EP.
Parameters	• hpcd: : Device instance • ep_addr: endpoint address • ep_kind: endpoint Kind USB_SNG_BUF: Single Buffer used USB_DBL_BUF: Double Buffer used • pmaaddress: EP address in The PMA: In case of single buffer endpoint this parameter is 16-bit value providing the address in PMA allocated to endpoint. In case of double buffer endpoint this parameter is a 32-bit value providing the endpoint buffer 0 address in the LSB part of 32-bit value and endpoint buffer 1 address in the MSB part of 32-bit value.
Return values	• :: status

33 HAL PWR Generic Driver

33.1 PWR Firmware driver registers structures

33.1.1 PWR_PVDTypeDef

Data Fields

- *uint32_t PVDLevel*
- *uint32_t Mode*

Field Documentation

- *uint32_t PWR_PVDTypeDef::PVDLevel*
PVDLevel: Specifies the PVD detection level. This parameter can be a value of [PWR_PVD_detection_level](#)
- *uint32_t PWR_PVDTypeDef::Mode*
Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [PWR_PVD_Mode](#)

33.2 PWR Firmware driver API description

33.2.1 Initialization and de-initialization functions

This section contains the following APIs:

- [HAL_PWR_DeInit\(\)](#)
- [HAL_PWR_EnableBkUpAccess\(\)](#)
- [HAL_PWR_DisableBkUpAccess\(\)](#)

33.2.2 Peripheral Control functions

Backup domain

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.

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in the `PWR_CR`).
- The PVD can use an external input analog voltage (`PVD_IN`) which is compared internally to `VREFINT`. The `PVD_IN` (PB7) has to be configured in Analog mode when `PWR_PVDLevel_7` is selected (PLS[2:0] = 111).

- A PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through `__HAL_PWR_PVD_EXTI_ENABLE_IT()` macro.
- The PVD is stopped in Standby mode.

WakeUp pin configuration

- WakeUp pin is used to wake up the system from Standby mode. This pin is forced in input pull-down configuration and is active on rising edges.
- There are two WakeUp pins: WakeUp Pin 1 on PA.00. WakeUp Pin 2 on PC.13. WakeUp Pin 3 on PE.06 .

Main and Backup Regulators configuration

Low Power modes configuration

The device features 5 low-power modes:

- Low power run mode: regulator in low power mode, limited clock frequency, limited number of peripherals running.
- Sleep mode: Cortex-M0+ core stopped, peripherals kept running.
- Low power sleep mode: Cortex-M0+ core stopped, limited clock frequency, limited number of peripherals running, regulator in low power mode.
- Stop mode: All clocks are stopped, regulator running, regulator in low power mode.
- Standby mode: VCORE domain powered off

Low power run mode

To further reduce the consumption when the system is in Run mode, the regulator can be configured in low power mode. In this mode, the system frequency should not exceed MSI frequency range1. In Low power run mode, all I/O pins keep the same state as in Run mode.

- Entry:
 - VCORE in range2
 - Decrease the system frequency not to exceed the frequency of MSI frequency range1.
 - The regulator is forced in low power mode using the `HAL_PWREx_EnableLowPowerRunMode()` function.
- Exit:
 - The regulator is forced in Main regulator mode using the `HAL_PWREx_DisableLowPowerRunMode()` function.
 - Increase the system frequency if needed.

Sleep mode

- Entry: The Sleep mode is entered by using the `HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFx)` functions with
 - `PWR_SLEEPENTRY_WFI`: enter SLEEP mode with WFI instruction
 - `PWR_SLEEPENTRY_WFE`: enter SLEEP mode with WFE instruction

- Exit:
 - Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode. If the WFE instruction was used to enter sleep mode, the MCU exits Sleep mode as soon as an event occurs.

Low power sleep mode

- Entry: The Low power sleep mode is entered by using the `HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFx)` functions with
 - `PWR_SLEEPENTRY_WFI`: enter SLEEP mode with WFI instruction
 - `PWR_SLEEPENTRY_WFE`: enter SLEEP mode with WFE instruction
- The Flash memory can be switched off by using the control bits (`SLEEP_PD` in the `FLASH_ACR` register). This reduces power consumption but increases the wake-up time.
- Exit:
 - If the WFI instruction was used to enter Low power sleep mode, any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Low power sleep mode. If the WFE instruction was used to enter Low power sleep mode, the MCU exits Sleep mode as soon as an event occurs.

Stop mode

The Stop mode is based on the Cortex-M0+ deepsleep mode combined with peripheral clock gating. The voltage regulator can be configured either in normal or low-power mode. In Stop mode, all clocks in the VCORE domain are stopped, the PLL, the MSI, the HSI and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. To get the lowest consumption in Stop mode, the internal Flash memory also enters low power mode. When the Flash memory is in power-down mode, an additional startup delay is incurred when waking up from Stop mode. To minimize the consumption In Stop mode, VREFINT, the BOR, PVD, and temperature sensor can be switched off before entering Stop mode. They can be switched on again by software after exiting Stop mode using the ULP bit in the `PWR_CR` register. In Stop mode, all I/O pins keep the same state as in Run mode.

- Entry: The Stop mode is entered using the `HAL_PWR_EnterSTOPMode` function with:
 - Main regulator ON.
 - Low Power regulator ON.
 - `PWR_SLEEPENTRY_WFI`: enter SLEEP mode with WFI instruction
 - `PWR_SLEEPENTRY_WFE`: enter SLEEP mode with WFE instruction
- Exit:
 - By issuing an interrupt or a wakeup event, the MSI or HSI16 RC oscillator is selected as system clock depending the bit `STOPWUCK` in the `RCC_CFGR` register

Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M0+ deepsleep mode, with the voltage regulator disabled. The VCORE domain is consequently powered off. The PLL, the MSI, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for the RTC registers, RTC backup registers and Standby circuitry. To minimize the consumption In Standby mode,

VREFINT, the BOR, PVD, and temperature sensor can be switched off before entering the Standby mode. They can be switched on again by software after exiting the Standby mode function.

- Entry:
 - The Standby mode is entered using the HAL_PWR_EnterSTANDBYMode() function.
- Exit:
 - WKUP pin rising edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

Auto-wakeup (AWU) from low-power mode

The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event, a time-stamp event, or a comparator event, without depending on an external interrupt (Auto-wakeup mode).

- RTC auto-wakeup (AWU) from the Stop mode
 - To wake up from the Stop mode with an RTC alarm event, it is necessary to:
 - Configure the EXTI Line 17 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
 - Enable the RTC Alarm Interrupt using the RTC_ITConfig() function
 - Configure the RTC to generate the RTC alarm using the RTC_SetAlarm() and RTC_AlarmCmd() functions.
 - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to:
 - Configure the EXTI Line 19 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
 - Enable the RTC Tamper or time stamp Interrupt using the RTC_ITConfig() function.
 - Configure the RTC to detect the tamper or time stamp event using the RTC_TimeStampConfig(), RTC_TamperTriggerConfig() and RTC_TamperCmd() functions.
 - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to:
 - Configure the EXTI Line 20 to be sensitive to rising edges (Interrupt or Event modes) using the EXTI_Init() function.
 - Enable the RTC WakeUp Interrupt using the RTC_ITConfig() function.
 - Configure the RTC to generate the RTC WakeUp event using the RTC_WakeUpClockConfig(), RTC_SetWakeUpCounter() and RTC_WakeUpCmd() functions.
- RTC auto-wakeup (AWU) from the Standby mode
 - To wake up from the Standby mode with an RTC alarm event, it is necessary to:
 - Enable the RTC Alarm Interrupt using the RTC_ITConfig() function.
 - Configure the RTC to generate the RTC alarm using the RTC_SetAlarm() and RTC_AlarmCmd() functions.
 - To wake up from the Standby mode with an RTC Tamper or time stamp event, it is necessary to:
 - Enable the RTC Tamper or time stamp Interrupt using the RTC_ITConfig() function.
 - Configure the RTC to detect the tamper or time stamp event using the RTC_TimeStampConfig(), RTC_TamperTriggerConfig() and RTC_TamperCmd() functions.
 - To wake up from the Standby mode with an RTC WakeUp event, it is necessary to:
 - Enable the RTC WakeUp Interrupt using the RTC_ITConfig() function

- Configure the RTC to generate the RTC WakeUp event using the `RTC_WakeUpClockConfig()`, `RTC_SetWakeUpCounter()` and `RTC_WakeUpCmd()` functions.
- Comparator auto-wakeup (AWU) from the Stop mode
 - To wake up from the Stop mode with an comparator 1 or comparator 2 wakeup event, it is necessary to:
 - Configure the EXTI Line 21 for comparator 1 or EXTI Line 22 for comparator 2 to be sensitive to the selected edges (falling, rising or falling and rising) (Interrupt or Event modes) using the `EXTI_Init()` function.
 - Configure the comparator to generate the event.

This section contains the following APIs:

- [`HAL\_PWR\_EnableBkUpAccess\(\)`](#)
- [`HAL\_PWR\_DisableBkUpAccess\(\)`](#)
- [`HAL\_PWR\_ConfigPVD\(\)`](#)
- [`HAL\_PWR\_EnablePVD\(\)`](#)
- [`HAL\_PWR\_DisablePVD\(\)`](#)
- [`HAL\_PWR\_EnableWakeUpPin\(\)`](#)
- [`HAL\_PWR\_DisableWakeUpPin\(\)`](#)
- [`HAL\_PWR\_EnterSLEEPMode\(\)`](#)
- [`HAL\_PWR\_EnterSTOPMode\(\)`](#)
- [`HAL\_PWR\_EnterSTANDBYMode\(\)`](#)
- [`HAL\_PWR\_EnableSleepOnExit\(\)`](#)
- [`HAL\_PWR\_DisableSleepOnExit\(\)`](#)
- [`HAL\_PWR\_EnableSEVOnPend\(\)`](#)
- [`HAL\_PWR\_DisableSEVOnPend\(\)`](#)
- [`HAL\_PWR\_PVD\_IRQHandler\(\)`](#)
- [`HAL\_PWR\_PVDCallback\(\)`](#)

33.2.3 HAL_PWR_DeInit

Function Name	<code>void HAL_PWR_DeInit (void)</code>
Function Description	Deinitializes the HAL PWR peripheral registers to their default reset values.
Return values	• None:

33.2.4 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).
Return values	• None:
Notes	• If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock, the Backup Domain Access should be kept enabled.

33.2.5 HAL_PWR_DisableBkUpAccess

Function Name	<code>void HAL_PWR_DisableBkUpAccess (void)</code>
Function Description	Disables access to the backup domain.
Return values	• None:

-
Notes	• Applies to RTC registers, RTC backup data registers. • If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock, the Backup Domain Access should be kept enabled.

33.2.6 HAL_PWR_EnableBkUpAccess

Function Name	void HAL_PWR_EnableBkUpAccess (void)
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers).
Return values	• None:
Notes	• If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock, the Backup Domain Access should be kept enabled.

33.2.7 HAL_PWR_DisableBkUpAccess

Function Name	void HAL_PWR_DisableBkUpAccess (void)
Function Description	Disables access to the backup domain.
Return values	• None:
Notes	• Applies to RTC registers, RTC backup data registers. • If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock, the Backup Domain Access should be kept enabled.

33.2.8 HAL_PWR_ConfigPVD

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

33.2.9 HAL_PWR_EnablePVD

Function Name	void HAL_PWR_EnablePVD (void)
Function Description	Enables the Power Voltage Detector(PVD).
Return values	• None:

33.2.10 HAL_PWR_DisablePVD

Function Name	void HAL_PWR_DisablePVD (void)
Function Description	Disables the Power Voltage Detector(PVD).
Return values	• None:

33.2.11 HAL_PWR_EnableWakeUpPin

Function Name	void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)
Function Description	Enables the WakeUp PINx functionality.
Parameters	WakeUpPinx: Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values: PWR_WAKEUP_PIN1PWR_WAKEUP_PIN2PWR_WAKEUP_PIN3 for stm32l07xxx and stm32l08xxx devices only.
Return values	None:

33.2.12 HAL_PWR_DisableWakeUpPin

Function Name	void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)
Function Description	Disables the WakeUp PINx functionality.
Parameters	WakeUpPinx: Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values: PWR_WAKEUP_PIN1PWR_WAKEUP_PIN2PWR_WAKEUP_PIN3 for stm32l07xxx and stm32l08xxx devices only.
Return values	None:

33.2.13 HAL_PWR_EnterSLEEPMode

Function Name	void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)
Function Description	Enters Sleep mode.
Parameters	Regulator: Specifies the regulator state in SLEEP mode. This parameter can be one of the following values: PWR_MAINREGULATOR_ON: SLEEP mode with regulator ONPWR_LOWPOWERREGULATOR_ON: SLEEP mode with low power regulator ON SLEEPEntry: Specifies if SLEEP mode is entered with WFI or WFE instruction. When WFI entry is used, tick interrupt have to be disabled if not desired as the interrupt wake up source. This parameter can be one of the following values: PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI 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.

33.2.14 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 instruction PWR_STOPENTRY_WFE: Enter Stop mode with WFE instruction

Return values

- **None:**

Notes

- In Stop mode, all I/O pins keep the same state as in Run mode.
- When exiting Stop mode by issuing an interrupt or a wakeup event, MSI or HSI16 RCoscillator is selected as system clock depending the bit STOPWUCK in the RCC_CFGR register.
- When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

33.2.15 HAL_PWR_EnterSTANDBYMode

Function Name

void HAL_PWR_EnterSTANDBYMode (void)

Function Description

Enters Standby mode.

Return values

- **None:**

Notes

- In Standby mode, all I/O pins are high impedance except for: Reset pad (still available) RTC_AF1 pin (PC13) if configured for tamper, time-stamp, RTC Alarm out, or RTC clock calibration out. RTC_AF2 pin (PC13) if configured for tamper. WKUP pin 1 (PA00) if enabled. WKUP pin 2 (PC13) if enabled. WKUP pin 3 (PE06) if enabled, for stm32l07xxx and stm32l08xxx devices only. WKUP pin 3 (PA02) if enabled, for stm32l031xx devices only.

33.2.16 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

- **None:**

Notes

- 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.

33.2.17 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	• None:
Notes	• Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

33.2.18 HAL_PWR_EnableSEVOnPend

Function Name	void HAL_PWR_EnableSEVOnPend (void)
Function Description	Enables CORTEX M0+ SEVONPEND bit.
Return values	• None:
Notes	• 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.

33.2.19 HAL_PWR_DisableSEVOnPend

Function Name	void HAL_PWR_DisableSEVOnPend (void)
Function Description	Disables CORTEX M0+ SEVONPEND bit.
Return values	• None:
Notes	• 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.

33.2.20 HAL_PWR_PVD_IRQHandler

Function Name	void HAL_PWR_PVD_IRQHandler (void)
Function Description	This function handles the PWR PVD interrupt request.
Return values	• None:
Notes	• This API should be called under the PVD_IRQHandler().

33.2.21 HAL_PWR_PVDCallback

Function Name	void HAL_PWR_PVDCallback (void)
Function Description	PWR PVD interrupt callback.
Return values	• None:

33.3 PWR Firmware driver defines

33.3.1 PWR

PWR Exported Macros

__HAL_PWR_VOLTAGESCALING_CONFIG	Description: • macros configure the main internal regulator output voltage. Parameters: • __REGULATOR__: specifies the

regulator output voltage to achieve a tradeoff between performance and power consumption when the device does not operate at the maximum frequency (refer to the datasheets for more details). This parameter can be one of the following values:

- PWR_REGULATOR_VOLTAGE_SCALE1: Regulator voltage output Scale 1 mode, System frequency up to 32 MHz.
- PWR_REGULATOR_VOLTAGE_SCALE2: Regulator voltage output Scale 2 mode, System frequency up to 16 MHz.
- PWR_REGULATOR_VOLTAGE_SCALE3: Regulator voltage output Scale 3 mode, System frequency up to 4.2 MHz

Return value:

- None

Description:

- Check PWR flag is set or not.

Parameters:

- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - PWR_FLAG_WU: Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm (Alarm B), 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.

`__HAL_PWR_GET_FLAG`

- PWR_FLAG_VREFINTRDY: Internal voltage reference (VREFINT) ready flag. This bit indicates the state of the internal voltage reference, VREFINT.
- PWR_FLAG_VOS: Voltage Scaling select flag. A delay is required for the internal regulator to be ready after the voltage range is changed. The VOSF bit indicates that the regulator has reached the voltage level defined with bits VOS of PWR_CR register.
- PWR_FLAG_REGLP: Regulator LP flag. When the MCU exits from Low power run mode, this bit stays at 1 until the regulator is ready in main mode. A polling on this bit is recommended to wait for the regulator main mode. This bit is reset by hardware when the regulator is ready.

Return value:

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

Description:

- Clear the PWR pending flags.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
 - PWR_FLAG_WU: Wake Up flag
 - PWR_FLAG_SB: StandBy flag

Description:

- Enable interrupt on PVD Exti Line 16.

Return value:

- None.

Description:

- Disable interrupt on PVD Exti Line 16.

Return value:

- None.

Description:

__HAL_PWR_CLEAR_FLAG

__HAL_PWR_PVD_EXTI_ENABLE_IT

__HAL_PWR_PVD_EXTI_DISABLE_IT

__HAL_PWR_PVD_EXTI_ENABLE_EVENT

	• Enable event on PVD Exti Line 16. Return value: • None.
<code>__HAL_PWR_PVD_EXTI_DISABLE_EVENT</code>	Description: • Disable event on PVD Exti Line 16. Return value: • None.
<code>__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE</code>	Description: • PVD EXTI line configuration: set falling edge trigger. Return value: • None.
<code>__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE</code>	Description: • Disable the PVD Extended Interrupt Falling Trigger. Return value: • None.
<code>__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE</code>	Description: • PVD EXTI line configuration: set rising edge trigger. Return value: • None.
<code>__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE</code>	Description: • Disable the PVD Extended Interrupt Rising Trigger. Return value: • None.
<code>__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE</code>	Description: • PVD EXTI line configuration: set rising & falling edge trigger. Return value: • None.
<code>__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE</code>	Description: • Disable the PVD Extended Interrupt Rising & Falling Trigger. Return value: • None.
<code>__HAL_PWR_PVD_EXTI_GET_FLAG</code>	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:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None.

__HAL_PWR_PVD_EXTI_CLEAR_FLAG

__HAL_PWR_PVD_EXTI_GENERATE_SWIT

__HAL_PWR_PVD_EXTI_GENERATE_SWIT

PWREx Flag Setting Time Out Value

PWR_FLAG_SETTING_DELAY_US

PWR Flag

PWR_FLAG_WU

PWR_FLAG_SB

PWR_FLAG_PVDO

PWR_FLAG_VREFINTRDY

PWR_FLAG_VOS

PWR_FLAG_REGLP

PVD detection level

PWR_PVDLEVEL_0

PWR_PVDLEVEL_1

PWR_PVDLEVEL_2

PWR_PVDLEVEL_3

PWR_PVDLEVEL_4

PWR_PVDLEVEL_5

PWR_PVDLEVEL_6

PWR_PVDLEVEL_7

PWR PVD Mode

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

PWR PVD Mode Mask

PVD_MODE_IT
PVD_MODE_EVT
PVD_RISING_EDGE
PVD_FALLING_EDGE

PWR Register alias address

PWR_WAKEUP_PIN1
PWR_WAKEUP_PIN2
PWR_WAKEUP_PIN3

PWR Regulator state in SLEEP/STOP mode

PWR_MAINREGULATOR_ON
PWR_LOWPOWERREGULATOR_ON

PWR Regulator Voltage Scale

PWR_REGULATOR_VOLTAGE_SCALE1
PWR_REGULATOR_VOLTAGE_SCALE2
PWR_REGULATOR_VOLTAGE_SCALE3
IS_PWR_VOLTAGE_SCALING_RANGE

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI
PWR_SLEEPENTRY_WFE

PWR STOP mode entry

PWR_STOPENTRY_WFI
PWR_STOPENTRY_WFE

34 HAL PWR Extension Driver

34.1 PWREx Firmware driver defines

34.1.1 PWREx

PWREx Exported Macros

`__HAL_PWR_FLASHWAKEUP_ENABLE` **Notes:**

- When entering low power mode (stop or standby only), if DS_EE_KOFF and RUN_PD of FLASH_ACR register are both set , the Flash memory will not be woken up when exiting from deep-sleep mode.

`__HAL_PWR_FLASHWAKEUP_DISABLE` **Notes:**

- When entering low power mode (stop or standby only), if DS_EE_KOFF and RUN_PD of FLASH_ACR register are both set , the Flash memory will not be woken up when exiting from deep-sleep mode.

35 HAL RCC Generic Driver

35.1 RCC Firmware driver registers structures

35.1.1 RCC_PLLInitTypeDef

Data Fields

- *uint32_t PLLState*
- *uint32_t PLLSource*
- *uint32_t PLLMUL*
- *uint32_t PLLDIV*

Field Documentation

- *uint32_t RCC_PLLInitTypeDef::PLLState*
The new state of the PLL. This parameter can be a value of [RCC_PLL_Config](#)
- *uint32_t RCC_PLLInitTypeDef::PLLSource*
RCC_PLLSource: PLL entry clock source. This parameter must be a value of [RCC_PLL_Clock_Source](#)
- *uint32_t RCC_PLLInitTypeDef::PLLMUL*
PLLMUL: Multiplication factor for PLL VCO output clock This parameter must of [RCC_PLLMultiplication_Factor](#)
- *uint32_t RCC_PLLInitTypeDef::PLLDIV*
PLLDIV: Division factor for main system clock (SYSCLK) This parameter must be a value of [RCC_PLLDivider_Factor](#)

35.1.2 RCC_OscInitTypeDef

Data Fields

- *uint32_t OscillatorType*
- *uint32_t HSEState*
- *uint32_t LSEState*
- *uint32_t HSISate*
- *uint32_t HSICalibrationValue*
- *uint32_t LSISate*
- *uint32_t HSI48State*
- *uint32_t MSISate*
- *uint32_t MSICalibrationValue*
- *uint32_t MSIClockRange*
- *RCC_PLLInitTypeDef PLL*

Field Documentation

- **`uint32_t RCC_OscInitTypeDef::OscillatorType`**
The oscillators to be configured. This parameter can be a value of [RCC_Oscillator_Type](#)
- **`uint32_t RCC_OscInitTypeDef::HSEState`**
The new state of the HSE. This parameter can be a value of [RCC_HSE_Config](#)
- **`uint32_t RCC_OscInitTypeDef::LSEState`**
The new state of the LSE. This parameter can be a value of [RCC_LSE_Config](#)
- **`uint32_t RCC_OscInitTypeDef::HSIState`**
The new state of the HSI. This parameter can be a value of [RCC_HSI_Config](#)
- **`uint32_t RCC_OscInitTypeDef::HSICalibrationValue`**
The calibration trimming value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F
- **`uint32_t RCC_OscInitTypeDef::LSIState`**
The new state of the LSI. This parameter can be a value of [RCC_LSI_Config](#)
- **`uint32_t RCC_OscInitTypeDef::HSI48State`**
The new state of the HSI48. This parameter can be a value of [RCC_HSI48_Config](#)
- **`uint32_t RCC_OscInitTypeDef::MSIState`**
The new state of the MSI. This parameter can be a value of [RCC_MSI_Config](#)
- **`uint32_t RCC_OscInitTypeDef::MSICalibrationValue`**
The calibration trimming value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- **`uint32_t RCC_OscInitTypeDef::MSIClockRange`**
The MSI frequency range. This parameter can be a value of [RCC_MSI_Clock_Range](#)
- **`RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL`**
PLL structure parameters

35.1.3 RCC_ClkInitTypeDef

Data Fields

- **`uint32_t ClockType`**
- **`uint32_t SYSCLKSource`**
- **`uint32_t AHBCLKDivider`**
- **`uint32_t APB1CLKDivider`**
- **`uint32_t APB2CLKDivider`**

Field Documentation

- **`uint32_t RCC_ClkInitTypeDef::ClockType`**
The clock to be configured. This parameter can be a value of [RCC_System_Clock_Type](#)
- **`uint32_t RCC_ClkInitTypeDef::SYSCLKSource`**
The clock source (SYSCLKS) used as system clock. This parameter can be a value of [RCC_System_Clock_Source](#)
- **`uint32_t RCC_ClkInitTypeDef::AHBCLKDivider`**
The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of [RCC_AHB_Clock_Source](#)
- **`uint32_t RCC_ClkInitTypeDef::APB1CLKDivider`**
The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of [RCC_APB1_APB2_Clock_Source](#)

- **`uint32_t RCC_ClkInitTypeDef::APB2CLKDivider`**
The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of [RCC_APB1_APB2_Clock_Source](#)

35.2 RCC Firmware driver API description

35.2.1 RCC specific features

After reset the device is running from MSI (2 MHz) with Flash 0 WS, all peripherals are off except internal SRAM, Flash and SW-DP.

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

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

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

35.2.2 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.
- If peripheral is mapped on AHB: the delay is 2 AHB clock cycle after the clock enable bit is set on the hardware register
- If peripheral is mapped on APB: the delay is 2 APB clock cycle after the clock enable bit is set on the hardware register

Possible Workarounds:

1. Enable the peripheral clock sometimes before the peripheral read/write register is required.
2. For AHB peripheral, insert two dummy read to the peripheral register.
3. For APB peripheral, insert a dummy read to the peripheral register.

35.2.3 Initialization and de-initialization functions

This section provide functions allowing to configure the internal/external clocks, PLL, CSS and MCO.

Internal/external clock and PLL configuration

1. HSI (high-speed internal), 16 MHz factory-trimmed RC used directly or through the PLL as System clock source.
2. MSI (multi-speed internal), multispeed low power RC (65.536 KHz to 4.194 MHz) used as System clock source.

3. LSI (low-speed internal), 37 KHz low consumption RC used as IWDG and/or RTC clock source.
4. HSE (high-speed external), 1 to 24 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
5. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
6. PLL (clocked by HSI or HSE), for System clock and USB (48 MHz).
7. CSS (Clock security system), once enable and if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to MSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.
8. MCO (microcontroller clock output), used to output SYSCLK, HSI, MSI, HSE, PLL, LSI or LSE clock (through a configurable prescaler) on PA8 pin.

System, AHB and APB busses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): MSI, HSI, HSE and PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on IOPORT, AHB bus (DMA,Flash...). APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "HAL_RCC_GetSysClockFreq()" function to retrieve the frequencies of these clocks. All the peripheral clocks are derived from the System clock (SYSCLK) except: I2S: the I2S clock can be derived from an external clock mapped on the I2S_CKIN pin. RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 16. You have to use __HAL_RCC_RTC_CONFIG() and __HAL_RCC_RTC_ENABLE() macros to configure this clock. USB FS, and RNG require a frequency equal to 48 MHz to work correctly. This clock is derived from the main PLL or HSI48 RC oscillator. IWDG clock which is always the LSI clock.
2. For the STM32L0xx devices, the maximum frequency of the SYSCLK, HCLK, APB1 and APB2 is 32 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly (see table below).

Table 17: Number of wait states (WS) vs CPU clock (HCLK) frequency

Latency	HCLK clock frequency (MHz)		
	voltage range: 1.65 to 3.6 V		voltage range: 2.0 to 3.6 V
	VCORE = 1.2 V	VCORE = 1.5 V	VCORE = 1.8 V
0W (1 CPU cycle)	$0 < \text{HCLK} \leq 2$	$0 < \text{HCLK} \leq 8$	$0 < \text{HCLK} \leq 16$
1WS (2 CPU cycles)	$2 < \text{HCLK} \leq 4$	$8 < \text{HCLK} \leq 16$	$16 < \text{HCLK} \leq 32$

This section contains the following APIs:

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

35.2.4 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_GetSysClockFreq\(\)](#)

- [HAL_RCC_GetHCLKFreq\(\)](#)
- [HAL_RCC_GetPCLK1Freq\(\)](#)
- [HAL_RCC_GetPCLK2Freq\(\)](#)
- [HAL_RCC_GetOscConfig\(\)](#)
- [HAL_RCC_GetClockConfig\(\)](#)
- [HAL_RCC_NMI_IRQHandler\(\)](#)
- [HAL_RCC_CSSCallback\(\)](#)

35.2.5 HAL_RCC_DeInit

Function Name	void HAL_RCC_DeInit (void)
Function Description	Resets the RCC clock configuration to the default reset state.
Return values	• None:
Notes	• The default reset state of the clock configuration is given below: MSI ON and used as system clock source (MSI range is not modified by this function, it keep the value configured by user application) HSI, HSE and PLL OFF AHB, APB1 and APB2 prescaler set to 1. CSS and MCO OFF All interrupts disabled • This function does not modify the configuration of the • -Peripheral clocks • -HSI48, LSI, LSE and RTC clocks

35.2.6 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	• RCC_OscInitStruct: pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.
Return values	• HAL: status
Notes	• The PLL is not disabled when used as system clock. • Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. • 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.

35.2.7 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	• RCC_ClkInitStruct: pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral.

Return values

Notes

- **FLatency:** FLASH Latency, this parameter depends on System Clock Frequency
- **None:**
- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function
- The MSI is used (enabled by hardware) as system clock source after startup from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).
- A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source will be ready.

35.2.8 HAL_RCC_MCOConfig

Function Name

void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)

Function Description

Selects the clock source to output on MCO pin.

Parameters

- **RCC_MCOx:** specifies the output direction for the clock source. For STM32L0xx family this parameter can have only one value: RCC_MCO1: Clock source to output on MCO pin(PA8).RCC_MCO2: Clock source to output on MCO pin(PA9).
- **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values:
RCC_MCO1SOURCE_NOCLOCK: No clock selected
RCC_MCO1SOURCE_SYCLK: System clock selected
RCC_MCO1SOURCE_HSI: HSI oscillator clock selected
RCC_MCO1SOURCE_MSI: MSI oscillator clock selected
RCC_MCO1SOURCE_HSE: HSE oscillator clock selected
RCC_MCO1SOURCE_PLLCLK: PLL clock selected
RCC_MCO1SOURCE_LSI: LSI clock selected
RCC_MCO1SOURCE_LSE: LSE clock selected and in STM32L052xx,STM32L053xx,STM32L062xx,STM32L063xx STM32L072xx,STM32L073xx,STM32L082xx,STM32L083xxRCC_MCO1SOURCE_HSI48: HSI48 clock selected
- **RCC_MCODiv:** specifies the MCO DIV. This parameter can be one of the following values: RCC_MCODIV_1: no division applied to MCO clock
RCC_MCODIV_2: division by 2 applied to MCO clock
RCC_MCODIV_4: division by 4 applied to MCO clock
RCC_MCODIV_8: division by 8 applied to MCO clock
RCC_MCODIV_16: division by 16 applied to MCO clock

Return values

- **None:**

Notes

- MCO pin should be configured in alternate function mode.

35.2.9 HAL_RCC_EnableCSS



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

35.2.10 HAL_RCC_GetSysClockFreq

Function Name	uint32_t HAL_RCC_GetSysClockFreq (void)
Function Description	Returns the SYSCLK frequency.
Return values	• SYSCLK: frequency
Notes	• The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source: • If SYSCLK source is MSI, function returns values based on MSI Value as defined by the MSI range. • If SYSCLK source is HSI, function returns values based on HSI_VALUE(*) • If SYSCLK source is HSE, function returns values based on HSE_VALUE(**) • If SYSCLK source is PLL, function returns values based on HSE_VALUE(**) or HSI_VALUE(*) multiplied/divided by the PLL factors. • (*) HSI_VALUE is a constant defined in stm32l0xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature. • (**) HSE_VALUE is a constant defined in stm32l0xx_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.

35.2.11 HAL_RCC_GetHCLKFreq

Function Name	uint32_t HAL_RCC_GetHCLKFreq (void)
Function Description	Returns the HCLK frequency.
Return values	• HCLK: frequency
Notes	• Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration

- based on this function will be incorrect.
- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function

35.2.12 HAL_RCC_GetPCLK1Freq

Function Name	uint32_t HAL_RCC_GetPCLK1Freq (void)
Function Description	Returns the PCLK1 frequency.
Return values	PCLK1: frequency
Notes	Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

35.2.13 HAL_RCC_GetPCLK2Freq

Function Name	uint32_t HAL_RCC_GetPCLK2Freq (void)
Function Description	Returns the PCLK2 frequency.
Return values	PCLK2: frequency
Notes	Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

35.2.14 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	RCC_OscInitStruct: pointer to an RCC_OscInitTypeDef structure that will be configured.
Return values	None:

35.2.15 HAL_RCC_GetClockConfig

Function Name	void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)
Function Description	Configures the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	RCC_ClkInitStruct: pointer to an RCC_ClkInitTypeDef structure that will be configured. pFLatency: Pointer on the Flash Latency.
Return values	None:

35.2.16 HAL_RCC_NMI_IRQHandler

Function Name	void HAL_RCC_NMI_IRQHandler (void)
Function Description	This function handles the RCC CSS interrupt request.

Return values	• None:
Notes	• This API should be called under the NMI_Handler().

35.2.17 HAL_RCC_CSSCallback

Function Name	void HAL_RCC_CSSCallback (void)
Function Description	RCC Clock Security System interrupt callback.
Return values	• None:

35.3 RCC Firmware driver defines

35.3.1 RCC

AHB Peripheral Clock Sleep Enable Disable

```

__HAL_RCC_CRC_CLK_SLEEP_ENABLE
__HAL_RCC_MIF_CLK_SLEEP_ENABLE
__HAL_RCC_SRAM_CLK_SLEEP_ENABLE
__HAL_RCC_DMA1_CLK_SLEEP_ENABLE
__HAL_RCC_CRC_CLK_SLEEP_DISABLE
__HAL_RCC_MIF_CLK_SLEEP_DISABLE
__HAL_RCC_SRAM_CLK_SLEEP_DISABLE
__HAL_RCC_DMA1_CLK_SLEEP_DISABLE

```

AHB Peripheral Clock Sleep Enabled or Disabled Status

```

__HAL_RCC_CRC_IS_CLK_SLEEP_ENABLED
__HAL_RCC_MIF_IS_CLK_SLEEP_ENABLED
__HAL_RCC_SRAM_IS_CLK_SLEEP_ENABLED
__HAL_RCC_DMA1_IS_CLK_SLEEP_ENABLED

```

RCC AHB Clock Source

```

RCC_SYSCLK_DIV1    SYSCLK not divided
RCC_SYSCLK_DIV2    SYSCLK divided by 2
RCC_SYSCLK_DIV4    SYSCLK divided by 4
RCC_SYSCLK_DIV8    SYSCLK divided by 8
RCC_SYSCLK_DIV16   SYSCLK divided by 16
RCC_SYSCLK_DIV64   SYSCLK divided by 64
RCC_SYSCLK_DIV128  SYSCLK divided by 128
RCC_SYSCLK_DIV256  SYSCLK divided by 256
RCC_SYSCLK_DIV512  SYSCLK divided by 512

```

AHB Peripheral Force Release Reset

```

__HAL_RCC_AHB_FORCE_RESET

```


__HAL_RCC_DMA1_FORCE_RESET
__HAL_RCC_MIF_FORCE_RESET
__HAL_RCC_CRC_FORCE_RESET
__HAL_RCC_AHB_RELEASE_RESET
__HAL_RCC_CRC_RELEASE_RESET
__HAL_RCC_DMA1_RELEASE_RESET
__HAL_RCC_MIF_RELEASE_RESET

AHB Peripheral Clock Enable Disable

__HAL_RCC_DMA1_CLK_ENABLE
__HAL_RCC_MIF_CLK_ENABLE
__HAL_RCC_CRC_CLK_ENABLE
__HAL_RCC_DMA1_CLK_DISABLE
__HAL_RCC_MIF_CLK_DISABLE
__HAL_RCC_CRC_CLK_DISABLE

AHB Peripheral Clock Enabled or Disabled Status

__HAL_RCC_DMA1_IS_CLK_ENABLED
__HAL_RCC_MIF_IS_CLK_ENABLED
__HAL_RCC_CRC_IS_CLK_ENABLED

RCC APB1 APB2 Clock Source

RCC_HCLK_DIV1	HCLK not divided
RCC_HCLK_DIV2	HCLK divided by 2
RCC_HCLK_DIV4	HCLK divided by 4
RCC_HCLK_DIV8	HCLK divided by 8
RCC_HCLK_DIV16	HCLK divided by 16

APB1 Peripheral Clock Enable Disable

__HAL_RCC_WWDG_CLK_ENABLE
__HAL_RCC_PWR_CLK_ENABLE
__HAL_RCC_WWDG_CLK_DISABLE
__HAL_RCC_PWR_CLK_DISABLE

APB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_WWDG_IS_CLK_ENABLED
__HAL_RCC_PWR_IS_CLK_ENABLED

APB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_WWDG_CLK_SLEEP_ENABLE
__HAL_RCC_PWR_CLK_SLEEP_ENABLE
__HAL_RCC_WWDG_CLK_SLEEP_DISABLE

`__HAL_RCC_PWR_CLK_SLEEP_DISABLE`

APB1 Peripheral Clock Sleep Enabled or Disabled Status

`__HAL_RCC_WWDG_IS_CLK_SLEEP_ENABLED`

`__HAL_RCC_PWR_IS_CLK_SLEEP_ENABLED`

APB1 Peripheral Force Release Reset

`__HAL_RCC_APB1_FORCE_RESET`

`__HAL_RCC_WWDG_FORCE_RESET`

`__HAL_RCC_PWR_FORCE_RESET`

`__HAL_RCC_APB1_RELEASE_RESET`

`__HAL_RCC_WWDG_RELEASE_RESET`

`__HAL_RCC_PWR_RELEASE_RESET`

APB2 Peripheral Clock Enable Disable

`__HAL_RCC_SYSCFG_CLK_ENABLE`

`__HAL_RCC_DBGMCU_CLK_ENABLE`

`__HAL_RCC_SYSCFG_CLK_DISABLE`

`__HAL_RCC_DBGMCU_CLK_DISABLE`

APB2 Peripheral Clock Enabled or Disabled Status

`__HAL_RCC_SYSCFG_IS_CLK_ENABLED`

`__HAL_RCC_DBGMCU_IS_CLK_ENABLED`

APB2 Peripheral Clock Sleep Enable Disable

`__HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE`

`__HAL_RCC_DBGMCU_CLK_SLEEP_ENABLE`

`__HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE`

`__HAL_RCC_DBGMCU_CLK_SLEEP_DISABLE`

APB2 Peripheral Clock Sleep Enabled or Disabled Status

`__HAL_RCC_SYSCFG_IS_CLK_SLEEP_ENABLED`

`__HAL_RCC_DBGMCU_IS_CLK_SLEEP_ENABLED`

APB2 Peripheral Force Release Reset

`__HAL_RCC_APB2_FORCE_RESET`

`__HAL_RCC_DBGMCU_FORCE_RESET`

`__HAL_RCC_SYSCFG_FORCE_RESET`

`__HAL_RCC_APB2_RELEASE_RESET`

`__HAL_RCC_DBGMCU_RELEASE_RESET`

`__HAL_RCC_SYSCFG_RELEASE_RESET`

RCC Backup Domain Reset

`__HAL_RCC_BACKUPRESET_FORCE`

Notes:

- This function resets the RTC peripheral

(including the backup registers) and the RTC clock source selection in RCC_CSR register. The BKPSRAM is not affected by this reset.

`__HAL_RCC_BACKUPRESET_RELEASE`

RCC Exported Macros

`__HAL_RCC_HSI_ENABLE`

Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

`__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. 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.

`__HAL_RCC_HSI_CONFIG`

Description:

- Macro to enable or disable the Internal High Speed oscillator (HSI).

Parameters:

- `__STATE__`: specifies the new state of the HSI. This parameter can be one of the following values:
 - `RCC_HSI_OFF`: turn OFF the HSI oscillator

- RCC_HSI_ON: turn ON the HSI oscillator
- RCC_HSI_DIV4: turn ON the HSI oscillator and divide it by 4

Notes:

- 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 to clock the PLL and/or system clock. HSI can not be stopped 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 stop the HSI. The HSI is stopped by hardware when entering STOP and STANDBY modes.
- When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_MSI_ENABLE

Notes:

- The MSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). MSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the MSI. After enabling the MSI, the application software should wait on MSIRDY flag to be set indicating that MSI clock is stable and can be used as system clock source. When the MSI is stopped, MSIRDY flag goes low after 6 MSI oscillator clock cycles.

__HAL_RCC_MSI_DISABLE

__HAL_RCC_MSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro Adjusts the Internal Multi Speed oscillator (MSI) calibration value.

Parameters:

- __MSICalibrationValue__: specifies the calibration trimming value. This parameter must be a number between 0 and 0xFF.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature

that influence the frequency of the internal MSI RC. Refer to the Application Note AN3300 for more details on how to calibrate the MSI.

`__HAL_RCC_MSI_RANGE_CONFIG`

Description:

- Macro to configures the Internal Multi Speed oscillator (MSI) clock range.

Parameters:

- `__RCC_MSIRange__`: specifies the MSI Clock range. This parameter must be one of the following values:
 - `RCC_MSIRANGE_0`: MSI clock is around 65.536 KHz
 - `RCC_MSIRANGE_1`: MSI clock is around 131.072 KHz
 - `RCC_MSIRANGE_2`: MSI clock is around 262.144 KHz
 - `RCC_MSIRANGE_3`: MSI clock is around 524.288 KHz
 - `RCC_MSIRANGE_4`: MSI clock is around 1.048 MHz
 - `RCC_MSIRANGE_5`: MSI clock is around 2.097 MHz (default after Reset or wake-up from STANDBY)
 - `RCC_MSIRANGE_6`: MSI clock is around 4.194 MHz

Notes:

- After restart from Reset or wakeup from STANDBY, the MSI clock is around 2.097 MHz. The MSI clock does not change after wake-up from STOP mode. The MSI clock range can be modified on the fly.

`__HAL_RCC_GET_MSI_RANGE`

Description:

- Macro to get the Internal Multi Speed oscillator (`__MSI__`) clock range in run mode.

Return value:

- MSI: clock range. This parameter must be one of the following values:
 - `RCC_MSIRANGE_0`: MSI clock is around 65.536 KHz
 - `RCC_MSIRANGE_1`: MSI clock is around 131.072 KHz
 - `RCC_MSIRANGE_2`: MSI clock is around 262.144 KHz
 - `RCC_MSIRANGE_3`: MSI clock is around 524.288 KHz
 - `RCC_MSIRANGE_4`: MSI clock is around 1.048 MHz

- RCC_MSIRANGE_5: MSI clock is around 2.097 MHz (default after Reset or wake-up from STANDBY)
- RCC_MSIRANGE_6: MSI clock is around 4.194 MHz

__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

__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_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. User should request a transition to LSE Off first and then LSE On or LSE Bypass. 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.

`__HAL_RCC_RTC_CLKPRESCALER`

Description:

- Configures or Get the RTC and LCD clock (`RTCCLK` / `LCDCLK`).

Parameters:

- `__RTCCLKSOURCE__`: specifies the RTC clock source. This parameter can be one of the following values:
 - `RCC_RTCCLKSOURCE_LSE`: LSE selected as RTC clock
 - `RCC_RTCCLKSOURCE_LSI`: LSI selected as RTC clock
 - `RCC_RTCCLKSOURCE_HSE_DIV2`: HSE divided by 2 selected as RTC clock
 - `RCC_RTCCLKSOURCE_HSE_DIV4`: HSE divided by 4 selected as RTC clock

- RCC_RTCCLKSOURCE_HSE_DIV8: HSE divided by 8 selected as RTC clock
- RCC_RTCCLKSOURCE_HSE_DIV16: HSE divided by 16 selected as RTC clock

Notes:

- As the RTC clock configuration bits are in the RTC domain and write access is denied to this domain after reset, you have to enable write access using PWR_RTCAccessCmd(ENABLE) function before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it cannot be changed unless the RTC is reset using RCC_RTCResetCmd function, or by a Power On Reset (POR). The RTC clock (RTCCLK) is used also to clock the LCD (LCDCLK).
- 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 HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

__HAL_RCC_RTC_CONFIG
 __HAL_RCC_GET_RTC_SOURCE
 __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 can not be disabled if it is used as system clock source. The main PLL is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL_DISABLE
 __HAL_RCC_PLL_CONFIG

Description:

- Macro to configure the main 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 to generate the PLL VCO clock This parameter must be one of the following values:
 - RCC_CFGR_PLLMUL3: PLLVCO = PLL clock entry x 3
 - RCC_CFGR_PLLMUL4: PLLVCO = PLL clock entry x 4
 - RCC_CFGR_PLLMUL6: PLLVCO = PLL clock entry x 6
 - RCC_CFGR_PLLMUL8: PLLVCO = PLL clock entry x 8
 - RCC_CFGR_PLLMUL12: PLLVCO = PLL clock entry x 12
 - RCC_CFGR_PLLMUL16: PLLVCO = PLL clock entry x 16
 - RCC_CFGR_PLLMUL24: PLLVCO = PLL clock entry x 24
 - RCC_CFGR_PLLMUL32: PLLVCO = PLL clock entry x 32
 - RCC_CFGR_PLLMUL48: PLLVCO = PLL clock entry x 48
- **__PLLDIV__**: specifies the PLL output clock division from PLL VCO clock This parameter must be one of the following values:
 - RCC_PLLDIV_2: PLL clock output = PLLVCO / 2
 - RCC_PLLDIV_3: PLL clock output = PLLVCO / 3
 - RCC_PLLDIV_4: PLL clock output = PLLVCO / 4

Notes:

- This function must be used only when the main PLL is disabled.
- The PLL VCO clock frequency must not exceed 96 MHz when the product is in Range 1, 48 MHz when the product is in Range 2 and 24 MHz when the product is in Range 3.

__HAL_RCC_GET_PLL_OSCSOURCE**Description:**

- Macro to get the oscillator used as PLL clock source.

Return value:

- The: oscillator used as PLL clock source.

The returned value can be one of the following:

- RCC_PLLSOURCE_HSI: HSI oscillator is used as PLL clock source.
- RCC_PLLSOURCE_HSE: HSE oscillator is used as PLL clock source.

__HAL_RCC_SYSCLK_CONFIG

Description:

- Macro to configure the system clock source.

Parameters:

- __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.

Return value:

- None

__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_MSI: MSI used as system clock.
 - RCC_SYSCLKSOURCE_STATUS_HSI: HSI used as system clock.
 - RCC_SYSCLKSOURCE_STATUS_HSE: HSE used as system clock.
 - RCC_SYSCLKSOURCE_STATUS_PLLCLK: PLL used as system clock.

__HAL_RCC_MCO1_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_CFGR_MCO_HSI`: HSI clock selected as MCO source
 - `RCC_CFGR_MCO_MSI`: MSI clock selected as MCO source
 - `RCC_CFGR_MCO_HSE`: HSE clock selected as MCO source
 - `RCC_CFGR_MCO_PLL`: PLL clock selected as MCO source
 - `RCC_CFGR_MCO_LSI`: LSI clock selected as MCO source
 - `RCC_CFGR_MCO_LSE`: LSE clock selected as MCO source
- `__MCO DIV__`: specifies the MCO clock prescaler. This parameter can be one of the following values:
 - `RCC_CFGR_MCO_PRE_1`: no division applied to MCO clock
 - `RCC_CFGR_MCO_PRE_2`: division by 2 applied to MCO clock
 - `RCC_CFGR_MCO_PRE_4`: division by 4 applied to MCO clock
 - `RCC_CFGR_MCO_PRE_8`: division by 8 applied to MCO clock
 - `RCC_CFGR_MCO_PRE_16`: division by 16 applied to MCO clock

RCC Flag`RCC_FLAG_HSIRDY``RCC_FLAG_HSIDIV``RCC_FLAG_MSIRDY``RCC_FLAG_HSERDY``RCC_FLAG_PLLRDY``RCC_FLAG_LSERDY``RCC_FLAG_LSECSS``RCC_FLAG_LSIRDY``RCC_FLAG_FWRST``RCC_FLAG_OBLRST``RCC_FLAG_PINRST``RCC_FLAG_PORRST``RCC_FLAG_SFTRST``RCC_FLAG_IWDGRST``RCC_FLAG_WWDGRST``RCC_FLAG_LPWRST`

RCC_FLAG_HSI48RDY

RCC Flags Interrupts Management`__HAL_RCC_ENABLE_IT`**Description:**

- Enable RCC interrupt (Perform Byte access to RCC_CIER[0:7] bits to enable the selected interrupts).

Parameters:

- `__INTERRUPT__`: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
 - `RCC_IT_LSIRDY`: LSI ready interrupt
 - `RCC_IT_LSERDY`: LSE ready interrupt
 - `RCC_IT_HSIIRDY`: HSI ready interrupt
 - `RCC_IT_HSERDY`: HSE ready interrupt
 - `RCC_IT_PLLRDY`: PLL ready interrupt
 - `RCC_IT_MSIRDY`: MSI ready interrupt
 - `RCC_IT_CSSLSE`: LSE CSS interrupt
 - `RCC_IT_HSI48RDY`: HSI48 ready interrupt

Notes:

- The CSS interrupt doesn't have an enable bit; once the CSS is enabled and if the HSE clock fails, the CSS interrupt occurs and an NMI is automatically generated. The NMI will be executed indefinitely, and since NMI has higher priority than any other IRQ (and main program) the application will be stacked in the NMI ISR unless the CSS interrupt pending bit is cleared.

`__HAL_RCC_DISABLE_IT`**Description:**

- Disable RCC interrupt (Perform Byte access to RCC_CIER[0:7] bits to disable the selected interrupts).

Parameters:

- `__INTERRUPT__`: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
 - `RCC_IT_LSIRDY`: LSI ready interrupt
 - `RCC_IT_LSERDY`: LSE ready interrupt
 - `RCC_IT_HSIIRDY`: HSI ready interrupt
 - `RCC_IT_HSERDY`: HSE ready interrupt
 - `RCC_IT_PLLRDY`: PLL ready interrupt
 - `RCC_IT_MSIRDY`: MSI ready interrupt
 - `RCC_IT_HSI48RDY`: HSI48 ready interrupt

- RCC_IT_CSSLSE: LSE CSS interrupt

Notes:

- The CSS interrupt doesn't have an enable bit; once the CSS is enabled and if the HSE clock fails, the CSS interrupt occurs and an NMI is automatically generated. The NMI will be executed indefinitely, and since NMI has higher priority than any other IRQ (and main program) the application will be stacked in the NMI ISR unless the CSS interrupt pending bit is cleared.

Description:

- Clear the RCC's interrupt pending bits (Perform Byte access to RCC_CIR[23:16] bits to clear the selected interrupt pending bits.

Parameters:

- `__INTERERRUPT__`: 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: PLL ready interrupt
 - RCC_IT_MSIRDY: MSI ready interrupt
 - RCC_IT_HSI48RDY: HSI48 ready interrupt
 - RCC_IT_CSSLSE: LSE CSS interrupt
 - RCC_IT_CSSHSE: Clock Security System interrupt

Description:

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

Parameters:

- `__INTERERRUPT__`: 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: PLL ready interrupt
 - RCC_IT_MSIRDY: MSI ready interrupt
 - RCC_IT_CSSLSE: LSE CSS interrupt
 - RCC_IT_CSSHSE: Clock Security

`__HAL_RCC_CLEAR_IT``__HAL_RCC_GET_IT`

System interrupt

Return value:

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

`__HAL_RCC_CLEAR_RESET_FLAGS`

The reset flags are: `RCC_FLAG_OBLRST`, `RCC_FLAG_PINRST`, `RCC_FLAG_PORRST`, `RCC_FLAG_SFTRST`, `RCC_FLAG_IWDGRST`, `RCC_FLAG_WWDGRST`, `RCC_FLAG_LPWRST`.

`__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_HSIDIV`: HSI clock divider flag
 - `RCC_FLAG_MSIRDY`: MSI oscillator clock ready
 - `RCC_FLAG_HSERDY`: HSE oscillator clock ready
 - `RCC_FLAG_PLLRDY`: PLL clock ready
 - `RCC_FLAG_LSECSS`: LSE oscillator clock CSS detected
 - `RCC_FLAG_LSERDY`: LSE oscillator clock ready
 - `RCC_FLAG_LSIRDY`: LSI oscillator clock ready
 - `RCC_FLAG_FWRST`: Firewall reset
 - `RCC_FLAG_OBLRST`: Option Byte Loader (OBL) reset
 - `RCC_FLAG_PINRST`: Pin reset
 - `RCC_FLAG_PORRST`: POR/PDR reset
 - `RCC_FLAG_SFTRST`: Software reset
 - `RCC_FLAG_IWDGRST`: Independent Watchdog reset
 - `RCC_FLAG_WWDGRST`: Window Watchdog reset
 - `RCC_FLAG_LPWRST`: Low Power reset

Return value:

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

RCC HSE Config`RCC_HSE_OFF`

RCC_HSE_ON

RCC_HSE_BYPASS

RCC HSI48 Configuration

RCC_HSI48_OFF

RCC_HSI48_ON

RCC HSI Configuration

RCC_HSI_OFF

RCC_HSI_ON

RCC_HSI_DIV4

RCC_HSI_OUTEN

RCC Interruptions

RCC_IT_LSIRDY

RCC_IT_LSERDY

RCC_IT_HSIRDY

RCC_IT_HSERDY

RCC_IT_PLLRDY

RCC_IT_MSIRDY

RCC_IT_CSSLSE

RCC_IT_CSSHSE

RCC_IT_HSI48RDY

IOPORT Peripheral Clock Enable Disable

__HAL_RCC_GPIOA_CLK_ENABLE

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_GPIOC_CLK_ENABLE

__HAL_RCC_GPIOH_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOH_CLK_DISABLE

IOPORT Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPIOA_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOB_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOC_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOH_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOA_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOB_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOC_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOH_CLK_SLEEP_DISABLE

IOPORT Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOH_IS_CLK_SLEEP_ENABLED

IOPORT Peripheral Force Release Reset

__HAL_RCC_IOP_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOH_FORCE_RESET

__HAL_RCC_IOP_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOH_RELEASE_RESET

IOPORT Peripheral Clock Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_ENABLED

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_GPIOC_IS_CLK_ENABLED

__HAL_RCC_GPIOH_IS_CLK_ENABLED

RCC LSE Config

RCC_LSE_OFF

RCC_LSE_ON

RCC_LSE_BYPASS

RCC LSI Config

RCC_LSI_OFF

RCC_LSI_ON

RCC_MSICALIBRATION_DEFAULT

RCC MCO1 Clock Source

RCC_MCO1SOURCE_NOCLOCK

RCC_MCO1SOURCE_SYSCLK

RCC_MCO1SOURCE_HSI

RCC_MCO1SOURCE_MSI

RCC_MCO1SOURCE_HSE
RCC_MCO1SOURCE_PLLCLK
RCC_MCO1SOURCE_LSI
RCC_MCO1SOURCE_LSE
RCC_MCO1SOURCE_HSI48

RCC MCO Prescaler

RCC_MCODIV_1
RCC_MCODIV_2
RCC_MCODIV_4
RCC_MCODIV_8
RCC_MCODIV_16

RCC MCO Index

RCC_MCO1
RCC_MCO2

RCC MSI Clock Range

RCC_MSIRANGE_0 MSI = 65.536 KHz
RCC_MSIRANGE_1 MSI = 131.072 KHz
RCC_MSIRANGE_2 MSI = 262.144 KHz
RCC_MSIRANGE_3 MSI = 524.288 KHz
RCC_MSIRANGE_4 MSI = 1.048 MHz
RCC_MSIRANGE_5 MSI = 2.097 MHz
RCC_MSIRANGE_6 MSI = 4.194 MHz

RCC MSI Config

RCC_MSI_OFF
RCC_MSI_ON
RCC_HSICALIBRATION_DEFAULT

Oscillator Type

RCC_OSCILLATORTYPE_NONE Oscillator configuration unchanged
RCC_OSCILLATORTYPE_HSE HSE to configure
RCC_OSCILLATORTYPE_HSI HSI to configure
RCC_OSCILLATORTYPE_LSE LSE to configure
RCC_OSCILLATORTYPE_LSI LSI to configure
RCC_OSCILLATORTYPE_MSI MSI to configure
RCC_OSCILLATORTYPE_HSI48

RCC PLL Dividers

RCC_PLLDIV_2

RCC_PLLDIV_3

RCC_PLLDIV_4

RCC PLL Multipliers

RCC_PLLMUL_3

RCC_PLLMUL_4

RCC_PLLMUL_6

RCC_PLLMUL_8

RCC_PLLMUL_12

RCC_PLLMUL_16

RCC_PLLMUL_24

RCC_PLLMUL_32

RCC_PLLMUL_48

RCC PLL Clock Source

RCC_PLLSOURCE_HSI

RCC_PLLSOURCE_HSE

RCC PLL Config

RCC_PLL_NONE

RCC_PLL_OFF

RCC_PLL_ON

RCC RTC Clock Configuration

__HAL_RCC_RTC_ENABLE **Notes:**

- These macros must be used only after the RTC clock source was selected.

__HAL_RCC_RTC_DISABLE

RCC RTC Clock Source

RCC_RTCCLKSOURCE_NO_CLK

RCC_RTCCLKSOURCE_LSE

RCC_RTCCLKSOURCE_LSI

RCC_RTCCLKSOURCE_HSE_DIV2

RCC_RTCCLKSOURCE_HSE_DIV4

RCC_RTCCLKSOURCE_HSE_DIV8

RCC_RTCCLKSOURCE_HSE_DIV16

RCC System Clock Source

RCC_SYSCLKSOURCE_MSI MSI selection as system clock

RCC_SYSCLKSOURCE_HSI HSI selection as system clock

RCC_SYSCLKSOURCE_HSE HSE selection as system clock

RCC_SYSCLKSOURCE_PLLCLK PLL selection as system clock

RCC System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_MSI	MSI used as system clock
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 System Clock Type

RCC_CLOCKTYPE_SYSCLK	SYSCLK to configure
RCC_CLOCKTYPE_HCLK	HCLK to configure
RCC_CLOCKTYPE_PCLK1	PCLK1 to configure
RCC_CLOCKTYPE_PCLK2	PCLK2 to configure

Timeout Values

RCC_DBP_TIMEOUT_VALUE
RCC_LSE_TIMEOUT_VALUE

36 HAL RCC Extension Driver

36.1 RCCEX Firmware driver registers structures

36.1.1 RCC_PeriphCLKInitTypeDef

Data Fields

- *uint32_t PeriphClockSelection*
- *uint32_t Usart1ClockSelection*
- *uint32_t Usart2ClockSelection*
- *uint32_t Lpuart1ClockSelection*
- *uint32_t I2c1ClockSelection*
- *uint32_t I2c3ClockSelection*
- *uint32_t RTCClockSelection*
- *uint32_t LCDClockSelection*
- *uint32_t UsbClockSelection*
- *uint32_t LptimClockSelection*

Field Documentation

- *uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection*
The Extended Clock to be configured. This parameter can be a value of [RCCEX_Periph_Clock_Selection](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection*
USART1 clock source This parameter can be a value of [RCCEX_USART1_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::Usart2ClockSelection*
USART2 clock source This parameter can be a value of [RCCEX_USART2_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::Lpuart1ClockSelection*
LPUART1 clock source This parameter can be a value of [RCCEX_LPUART1_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection*
I2C1 clock source This parameter can be a value of [RCCEX_I2C1_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::I2c3ClockSelection*
I2C3 clock source This parameter can be a value of [RCCEX_I2C3_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection*
Specifies RTC Clock Prescalers Selection This parameter can be a value of [RCC_RTC_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::LCDClockSelection*
specifies the LCD clock source. This parameter can be a value of [RCC_RTC_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection*
Specifies USB and RNG Clock Selection This parameter can be a value of [RCCEX_USB_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::LptimClockSelection*
LPTIM1 clock source This parameter can be a value of [RCCEX_LPTIM1_Clock_Source](#)

36.1.2 RCC_CRSSynchroTypeDef

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_CRSSynchroTypeDef::Prescaler***
Specifies the division factor of the SYNC signal. This parameter can be a value of [RCCEX_CRS_SynchroDivider](#)
- ***uint32_t RCC_CRSSynchroTypeDef::Source***
Specifies the SYNC signal source. This parameter can be a value of [RCCEX_CRS_SynchroSource](#)
- ***uint32_t RCC_CRSSynchroTypeDef::Polarity***
Specifies the input polarity for the SYNC signal source. This parameter can be a value of [RCCEX_CRS_SynchroPolarity](#)
- ***uint32_t RCC_CRSSynchroTypeDef::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 [RCCEX_CRS_ReloadValueDefault](#) .
- ***uint32_t RCC_CRSSynchroTypeDef::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 [RCCEX_CRS_ErrorLimitDefault](#)
- ***uint32_t RCC_CRSSynchroTypeDef::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 [RCCEX_CRS_HSI48CalibrationDefault](#)

36.1.3 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 [RCCEX_CRS_FreqErrorDirection](#)

36.2 RCCEX Firmware driver API description

36.2.1 Extended 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_DeInit\(\)](#)
- [HAL_RCCEX_PeriphCLKConfig\(\)](#)
- [HAL_RCCEX_GetPeriphCLKConfig\(\)](#)
- [HAL_RCCEX_GetPeriphCLKFreq\(\)](#)
- [HAL_RCCEX_EnableLSECSS\(\)](#)
- [HAL_RCCEX_DisableLSECSS\(\)](#)
- [HAL_RCCEX_EnableLSECSS_IT\(\)](#)
- [HAL_RCCEX_LSECSS_IRQHandler\(\)](#)
- [HAL_RCCEX_LSECSS_Callback\(\)](#)
- [HAL_RCCEX_CRSCConfig\(\)](#)
- [HAL_RCCEX_CRSSoftwareSynchronizationGenerate\(\)](#)
- [HAL_RCCEX_CRSCGetSynchronizationInfo\(\)](#)
- [HAL_RCCEX_CRSCWaitSynchronization\(\)](#)
- [HAL_RCCEX_EnableHSI48_VREFINT\(\)](#)
- [HAL_RCCEX_DisableHSI48_VREFINT\(\)](#)

36.2.2 HAL_RCC_DeInit

Function Name	<code>void HAL_RCC_DeInit (void)</code>
Function Description	Resets the RCC clock configuration to the default reset state.
Return values	• None:
Notes	• The default reset state of the clock configuration is given below: MSI ON and used as system clock source (MSI range is not modified by this function, it keep the value configured by user application) HSI, HSE and PLL OFFAHB, APB1 and APB2 prescaler set to 1. CSS and MCO OFF All interrupts disabled • This function does not modify the configuration of the • -Peripheral clocks • -HSI48, LSI, LSE and RTC clocks

36.2.3 HAL_RCCEX_PeriphCLKConfig

Function Name	HAL_StatusTypeDef HAL_RCCEX_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)
Function Description	Initializes the RCC extended peripherals clocks.
Parameters	PeriphClkInit: pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks(USART1,USART2, LPUART1, I2C1, I2C3, RTC, USB/RNG and LPTIM1 clocks).
Return values	HAL: status
Notes	Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.

36.2.4 HAL_RCCEX_GetPeriphCLKConfig

Function Name	void HAL_RCCEX_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)
Function Description	Get the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	PeriphClkInit: pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks(USART1,USART2, LPUART1, I2C1, I2C3, RTC, USB/RNG and LPTIM1 clocks).
Return values	None:

36.2.5 HAL_RCCEX_GetPeriphCLKFreq

Function Name	uint32_t HAL_RCCEX_GetPeriphCLKFreq (uint32_t PeriphClk)
Function Description	Return the peripheral clock frequency for some peripherals.
Parameters	PeriphClk: Peripheral clock identifier This parameter can be one of the following values: RCC_PERIPHCLK_RTC: RTC peripheral clockRCC_PERIPHCLK_LCD: LCD peripheral clock (*)RCC_PERIPHCLK_USB: USB or RNG peripheral clock (*)RCC_PERIPHCLK_USART1: USART1 peripheral clock (*)RCC_PERIPHCLK_USART2: USART2 peripheral clockRCC_PERIPHCLK_LPUART1: LPUART1 peripheral clockRCC_PERIPHCLK_I2C1: I2C1 peripheral clockRCC_PERIPHCLK_I2C2: I2C2 peripheral clock (*)RCC_PERIPHCLK_I2C3: I2C3 peripheral clock (*)
Return values	Frequency: in Hz (0: means that no available frequency for the peripheral)
Notes	- Return 0 if peripheral clock identifier not managed by this API (*) means that this peripheral is not present on all the STM32F0xx devices

36.2.6 HAL_RCCEX_EnableLSECSS

Function Name	void HAL_RCCEX_EnableLSECSS (void)

Function Description Enables the LSE Clock Security System.

Return values

- None:

36.2.7 HAL_RCCEX_DisableLSECSS

Function Name **void HAL_RCCEX_DisableLSECSS (void)**

Function Description Disables the LSE Clock Security System.

Return values

- None:

36.2.8 HAL_RCCEX_EnableLSECSS_IT

Function Name **void HAL_RCCEX_EnableLSECSS_IT (void)**

Function Description Enable the LSE Clock Security System IT & corresponding EXTI line.

Return values

- None:

Notes

- LSE Clock Security System IT is mapped on RTC EXTI line 19

36.2.9 HAL_RCCEX_LSECSS_IRQHandler

Function Name **void HAL_RCCEX_LSECSS_IRQHandler (void)**

Function Description Handle the RCC LSE Clock Security System interrupt request.

Return values

- None:

36.2.10 HAL_RCCEX_LSECSS_Callback

Function Name **void HAL_RCCEX_LSECSS_Callback (void)**

Function Description RCCEX LSE Clock Security System interrupt callback.

Return values

- none:

36.2.11 HAL_RCCEX_CRSCConfig

Function Name **void HAL_RCCEX_CRSCConfig (RCC_CRSSInitTypeDef * pInit)**

Function Description Start automatic synchronization using polling mode.

Parameters

- **pInit:** Pointer on RCC_CRSSInitTypeDef structure

Return values

- None:

36.2.12 HAL_RCCEX_CRSSoftwareSynchronizationGenerate

Function Name **void HAL_RCCEX_CRSSoftwareSynchronizationGenerate (void)**

Function Description Generate the software synchronization event.

Return values

- None:

36.2.13 HAL_RCCEX_CRSGetSynchronizationInfo

Function Name	void HAL_RCCEX_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)
Function Description	Function to return synchronization info.
Parameters	• pSynchroInfo: Pointer on RCC_CRSSynchroInfoTypeDef structure
Return values	• None:

36.2.14 HAL_RCCEX_CRSSWaitSynchronization

Function Name	uint32_t HAL_RCCEX_CRSSWaitSynchronization (uint32_t Timeout)
Function Description	This function handles CRS Synchronization Timeout.
Parameters	• Timeout: Duration of the timeout
Return values	• Combination: of Synchronization status This parameter can be a combination of the following values: RCC_CRCS_TIMEOUTRCC_CRCS_SYNCOKRCC_CRCS_SYNCWARMRCC_CRCS_SYNCERRRCC_CRCS_SYNCMISSRCC_CRCS_TRIMOV
Notes	• Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency. • If Timeout set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.

36.2.15 HAL_RCCEX_EnableHSI48_VREFINT

Function Name	void HAL_RCCEX_EnableHSI48_VREFINT (void)
Function Description	Enables Vrefint for the HSI48.
Return values	• None:
Notes	• This is functional only if the LOCK is not set

36.2.16 HAL_RCCEX_DisableHSI48_VREFINT

Function Name	void HAL_RCCEX_DisableHSI48_VREFINT (void)
Function Description	Disables the Vrefint for the HSI48.
Return values	• None:
Notes	• This is functional only if the LOCK is not set

36.3 RCCEX Firmware driver defines

36.3.1 RCCEX

AHB Peripheral Clock Sleep Enable Disable

`__HAL_RCC_TSC_CLK_SLEEP_ENABLE`

__HAL_RCC_RNG_CLK_SLEEP_ENABLE
__HAL_RCC_TSC_CLK_SLEEP_DISABLE
__HAL_RCC_RNG_CLK_SLEEP_DISABLE
__HAL_RCC_AES_CLK_SLEEP_ENABLE
__HAL_RCC_AES_CLK_SLEEP_DISABLE

AHB Peripheral Force Release Reset

__HAL_RCC_AES_FORCE_RESET
__HAL_RCC_AES_RELEASE_RESET
__HAL_RCC_TSC_FORCE_RESET
__HAL_RCC_TSC_RELEASE_RESET
__HAL_RCC_RNG_FORCE_RESET
__HAL_RCC_RNG_RELEASE_RESET

APB1 Peripheral Clock Enable Disable

__HAL_RCC_USB_CLK_ENABLE
__HAL_RCC_USB_CLK_DISABLE
__HAL_RCC_CRS_CLK_ENABLE
__HAL_RCC_CRS_CLK_DISABLE
__HAL_RCC_LCD_CLK_ENABLE
__HAL_RCC_LCD_CLK_DISABLE
__HAL_RCC_TIM2_CLK_ENABLE
__HAL_RCC_TIM3_CLK_ENABLE
__HAL_RCC_TIM6_CLK_ENABLE
__HAL_RCC_TIM7_CLK_ENABLE
__HAL_RCC_SPI2_CLK_ENABLE
__HAL_RCC_USART2_CLK_ENABLE
__HAL_RCC_USART4_CLK_ENABLE
__HAL_RCC_USART5_CLK_ENABLE
__HAL_RCC_LPUART1_CLK_ENABLE
__HAL_RCC_I2C1_CLK_ENABLE
__HAL_RCC_I2C2_CLK_ENABLE
__HAL_RCC_I2C3_CLK_ENABLE
__HAL_RCC_DAC_CLK_ENABLE
__HAL_RCC_LPTIM1_CLK_ENABLE
__HAL_RCC_TIM2_CLK_DISABLE
__HAL_RCC_TIM3_CLK_DISABLE
__HAL_RCC_TIM6_CLK_DISABLE

__HAL_RCC_TIM7_CLK_DISABLE
__HAL_RCC_SPI2_CLK_DISABLE
__HAL_RCC_USART2_CLK_DISABLE
__HAL_RCC_USART4_CLK_DISABLE
__HAL_RCC_USART5_CLK_DISABLE
__HAL_RCC_LPUART1_CLK_DISABLE
__HAL_RCC_I2C1_CLK_DISABLE
__HAL_RCC_I2C2_CLK_DISABLE
__HAL_RCC_I2C3_CLK_DISABLE
__HAL_RCC_DAC_CLK_DISABLE
__HAL_RCC_LPTIM1_CLK_DISABLE

APB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM2_CLK_SLEEP_ENABLE
__HAL_RCC_TIM3_CLK_SLEEP_ENABLE
__HAL_RCC_TIM6_CLK_SLEEP_ENABLE
__HAL_RCC_TIM7_CLK_SLEEP_ENABLE
__HAL_RCC_SPI2_CLK_SLEEP_ENABLE
__HAL_RCC_USART2_CLK_SLEEP_ENABLE
__HAL_RCC_USART4_CLK_SLEEP_ENABLE
__HAL_RCC_USART5_CLK_SLEEP_ENABLE
__HAL_RCC_LPUART1_CLK_SLEEP_ENABLE
__HAL_RCC_I2C1_CLK_SLEEP_ENABLE
__HAL_RCC_I2C2_CLK_SLEEP_ENABLE
__HAL_RCC_I2C3_CLK_SLEEP_ENABLE
__HAL_RCC_DAC_CLK_SLEEP_ENABLE
__HAL_RCC_LPTIM1_CLK_SLEEP_ENABLE
__HAL_RCC_TIM2_CLK_SLEEP_DISABLE
__HAL_RCC_TIM3_CLK_SLEEP_DISABLE
__HAL_RCC_TIM6_CLK_SLEEP_DISABLE
__HAL_RCC_TIM7_CLK_SLEEP_DISABLE
__HAL_RCC_SPI2_CLK_SLEEP_DISABLE
__HAL_RCC_USART2_CLK_SLEEP_DISABLE
__HAL_RCC_USART4_CLK_SLEEP_DISABLE
__HAL_RCC_USART5_CLK_SLEEP_DISABLE
__HAL_RCC_LPUART1_CLK_SLEEP_DISABLE
__HAL_RCC_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C2_CLK_SLEEP_DISABLE
__HAL_RCC_I2C3_CLK_SLEEP_DISABLE
__HAL_RCC_DAC_CLK_SLEEP_DISABLE
__HAL_RCC_LPTIM1_CLK_SLEEP_DISABLE
__HAL_RCC_USB_CLK_SLEEP_ENABLE
__HAL_RCC_USB_CLK_SLEEP_DISABLE
__HAL_RCC_CR2_CLK_SLEEP_ENABLE
__HAL_RCC_CR2_CLK_SLEEP_DISABLE
__HAL_RCC_LCD_CLK_SLEEP_ENABLE
__HAL_RCC_LCD_CLK_SLEEP_DISABLE

APB1 Peripheral Force Release Reset

__HAL_RCC_TIM2_FORCE_RESET
__HAL_RCC_TIM3_FORCE_RESET
__HAL_RCC_TIM6_FORCE_RESET
__HAL_RCC_TIM7_FORCE_RESET
__HAL_RCC_LPTIM1_FORCE_RESET
__HAL_RCC_I2C1_FORCE_RESET
__HAL_RCC_I2C2_FORCE_RESET
__HAL_RCC_I2C3_FORCE_RESET
__HAL_RCC_USART2_FORCE_RESET
__HAL_RCC_USART4_FORCE_RESET
__HAL_RCC_USART5_FORCE_RESET
__HAL_RCC_LPUART1_FORCE_RESET
__HAL_RCC_SPI2_FORCE_RESET
__HAL_RCC_DAC_FORCE_RESET
__HAL_RCC_TIM2_RELEASE_RESET
__HAL_RCC_TIM3_RELEASE_RESET
__HAL_RCC_TIM6_RELEASE_RESET
__HAL_RCC_TIM7_RELEASE_RESET
__HAL_RCC_LPTIM1_RELEASE_RESET
__HAL_RCC_I2C1_RELEASE_RESET
__HAL_RCC_I2C2_RELEASE_RESET
__HAL_RCC_I2C3_RELEASE_RESET
__HAL_RCC_USART2_RELEASE_RESET
__HAL_RCC_USART4_RELEASE_RESET
__HAL_RCC_USART5_RELEASE_RESET

__HAL_RCC_LPUART1_RELEASE_RESET
__HAL_RCC_SPI2_RELEASE_RESET
__HAL_RCC_DAC_RELEASE_RESET
__HAL_RCC_USB_FORCE_RESET
__HAL_RCC_USB_RELEASE_RESET
__HAL_RCC_CR2_FORCE_RESET
__HAL_RCC_CR2_RELEASE_RESET
__HAL_RCC_LCD_FORCE_RESET
__HAL_RCC_LCD_RELEASE_RESET

APB2 Peripheral Clock Enable Disable

__HAL_RCC_TIM21_CLK_ENABLE
__HAL_RCC_TIM22_CLK_ENABLE
__HAL_RCC_ADC1_CLK_ENABLE
__HAL_RCC_SPI1_CLK_ENABLE
__HAL_RCC_USART1_CLK_ENABLE
__HAL_RCC_TIM21_CLK_DISABLE
__HAL_RCC_TIM22_CLK_DISABLE
__HAL_RCC_ADC1_CLK_DISABLE
__HAL_RCC_SPI1_CLK_DISABLE
__HAL_RCC_USART1_CLK_DISABLE
__HAL_RCC_FIREWALL_CLK_ENABLE
__HAL_RCC_FIREWALL_CLK_DISABLE

APB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM21_CLK_SLEEP_ENABLE
__HAL_RCC_TIM22_CLK_SLEEP_ENABLE
__HAL_RCC_ADC1_CLK_SLEEP_ENABLE
__HAL_RCC_SPI1_CLK_SLEEP_ENABLE
__HAL_RCC_USART1_CLK_SLEEP_ENABLE
__HAL_RCC_TIM21_CLK_SLEEP_DISABLE
__HAL_RCC_TIM22_CLK_SLEEP_DISABLE
__HAL_RCC_ADC1_CLK_SLEEP_DISABLE
__HAL_RCC_SPI1_CLK_SLEEP_DISABLE
__HAL_RCC_USART1_CLK_SLEEP_DISABLE
__HAL_RCC_I2C1_CONFIG

Description:

- Macro to configure the I2C1 clock

(I2C1CLK).

Parameters:

- `__I2C1_CLKSOURCE__`: specifies the I2C1 clock source. This parameter can be one of the following values:
 - `RCC_I2C1CLKSOURCE_PCLK1`: PCLK1 selected as I2C1 clock
 - `RCC_I2C1CLKSOURCE_HSI`: HSI selected as I2C1 clock
 - `RCC_I2C1CLKSOURCE_SYSC_LK`: System Clock selected as I2C1 clock

Return value:

- None

`__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_PCLK1`: PCLK1 selected as I2C1 clock
 - `RCC_I2C1CLKSOURCE_HSI`: HSI selected as I2C1 clock
 - `RCC_I2C1CLKSOURCE_SYSC_LK`: System Clock selected as I2C1 clock

`__HAL_RCC_I2C3_CONFIG`

Description:

- Macro to configure the I2C3 clock (I2C3CLK).

Parameters:

- `__I2C3_CLKSOURCE__`: specifies the I2C3 clock source. This parameter can be one of the following values:
 - `RCC_I2C3CLKSOURCE_PCLK1`: PCLK1 selected as I2C3 clock
 - `RCC_I2C3CLKSOURCE_HSI`: HSI selected as I2C3 clock
 - `RCC_I2C3CLKSOURCE_SYSC_LK`: System Clock selected as I2C3 clock

Return value:

- None

`__HAL_RCC_GET_I2C3_SOURCE`

Description:

- Macro to get the I2C3 clock source.

Return value:

`__HAL_RCC_USART1_CONFIG`

- The: clock source can be one of the following values:
 - `RCC_I2C3CLKSOURCE_PCLK1`: PCLK1 selected as I2C3 clock
 - `RCC_I2C3CLKSOURCE_HSI`: HSI selected as I2C3 clock
 - `RCC_I2C3CLKSOURCE_SYSC`: LK: System Clock selected as I2C3 clock

Description:

- Macro to configure the USART1 clock (USART1CLK).

Parameters:

- `__USART1_CLKSOURCE__`: specifies the USART1 clock source. This parameter can be one of the following values:
 - `RCC_USART1CLKSOURCE_PCLK2`: PCLK2 selected as USART1 clock
 - `RCC_USART1CLKSOURCE_HSI`: HSI selected as USART1 clock
 - `RCC_USART1CLKSOURCE_SYSCLK`: System Clock selected as USART1 clock
 - `RCC_USART1CLKSOURCE_LSE`: LSE selected as USART1 clock

Return value:

- None

`__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_PCLK2`: PCLK2 selected as USART1 clock
 - `RCC_USART1CLKSOURCE_HSI`: HSI selected as USART1 clock
 - `RCC_USART1CLKSOURCE_SYSCLK`: System Clock selected as USART1 clock
 - `RCC_USART1CLKSOURCE_LSE`: LSE selected as USART1 clock

__HAL_RCC_USART2_CONFIG**Description:**

- Macro to configure the USART2 clock (USART2CLK).

Parameters:

- __USART2_CLKSOURCE__**: 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

Return value:

- None

__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_LPUART1_CONFIG**Description:**

- Macro to configure the LPUART1 clock (LPUART1CLK).

Parameters:

- __LPUART1_CLKSOURCE__**: specifies the LPUART1 clock source. This parameter can be one of the following values:

- RCC_LPUART1CLKSOURCE_P
CLK1: PCLK1 selected as LPUART1 clock
- RCC_LPUART1CLKSOURCE_H
SI: HSI selected as LPUART1 clock
- RCC_LPUART1CLKSOURCE_S
YCLK: System Clock selected as LPUART1 clock
- RCC_LPUART1CLKSOURCE_L
SE: LSE selected as LPUART1 clock

Return value:

- None

__HAL_RCC_GET_LPUART1_SOURCE**Description:**

- Macro to get the LPUART1 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPUART1CLKSOURCE_P
CLK1: PCLK1 selected as LPUART1 clock
 - RCC_LPUART1CLKSOURCE_H
SI: HSI selected as LPUART1 clock
 - RCC_LPUART1CLKSOURCE_S
YCLK: System Clock selected as LPUART1 clock
 - RCC_LPUART1CLKSOURCE_L
SE: LSE selected as LPUART1 clock

__HAL_RCC_LPTIM1_CONFIG**Description:**

- Macro to configure the LPTIM1 clock (LPTIM1CLK).

Parameters:

- **__LPTIM1_CLKSOURCE__**: specifies the LPTIM1 clock source. This parameter can be one of the following values:
 - RCC_LPTIM1CLKSOURCE_PC
LK: PCLK selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_LSI
: HSI selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_HSI
: LSI selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_LS
E : LSE selected as LPTIM1

clock

`__HAL_RCC_GET_LPTIM1_SOURCE`**Return value:**

- None

Description:

- Macro to get the LPTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
 - `RCC_LPTIM1CLKSOURCE_PCLK`: PCLK selected as LPUART1 clock
 - `RCC_LPTIM1CLKSOURCE_LSI`: HSI selected as LPUART1 clock
 - `RCC_LPTIM1CLKSOURCE_HSI`: System Clock selected as LPUART1 clock
 - `RCC_LPTIM1CLKSOURCE_LSE`: LSE selected as LPUART1 clock

`__HAL_RCC_USB_CONFIG`**Description:**

- Macro to configure the USB clock (USBCLK).

Parameters:

- `__USBCLKSource__`: specifies the USB clock source. This parameter can be one of the following values:
 - `RCC_USBCLKSOURCE_HSI48`: HSI48 selected as USB clock
 - `RCC_USBCLKSOURCE_PLL`: PLL Clock selected as USB clock

`__HAL_RCC_GET_USB_SOURCE`**Description:**

- Macro to get the USB clock source.

Return value:

- The: clock source can be one of the following values:
 - `RCC_USBCLKSOURCE_HSI48`: HSI48 selected as USB clock
 - `RCC_USBCLKSOURCE_PLL`: PLL Clock selected as USB clock

`__HAL_RCC_RNG_CONFIG`**Description:**

- Macro to configure the RNG clock (RNGCLK).

__HAL_RCC_GET_RNG_SOURCE

__HAL_RCC_HSI48M_CONFIG

__HAL_RCC_GET_HSI48M_SOURCE

Parameters:

- `__RNGCLKSource__`: specifies the USB clock source. This parameter can be one of the following values:
 - `RCC_RNGCLKSOURCE_HSI48`: HSI48 selected as RNG clock
 - `RCC_RNGCLKSOURCE_PLLCLK`: PLL Clock selected as RNG clock

Description:

- Macro to get the RNG clock source.

Return value:

- The clock source can be one of the following values:
 - `RCC_RNGCLKSOURCE_HSI48`: HSI48 selected as RNG clock
 - `RCC_RNGCLKSOURCE_PLLCLK`: PLL Clock selected as RNG clock

Description:

- macro to select the HSI48M clock source

Parameters:

- `__HSI48MCLKSource__`: specifies the HSI48M clock source dedicated for USB and RNG peripherals. This parameter can be one of the following values:
 - `RCC_HSI48M_PLL`: A dedicated 48MHZ PLL output.
 - `RCC_HSI48M_HSI48`: 48MHZ issued from internal HSI48 oscillator.

Notes:

- This macro can be replaced by either `__HAL_RCC_RNG_CONFIG` or `__HAL_RCC_USB_CONFIG` to configure respectively RNG or USB clock sources.

Description:

- macro to get the HSI48M clock source.

Return value:

- The clock source can be one of the following values:
 - `RCC_HSI48M_PLL`: A dedicated 48MHZ PLL output.

- RCC_HSI48M_HSI48: 48MHZ issued from internal HSI48 oscillator.

Notes:

- This macro can be replaced by either `__HAL_RCC_GET_RNG_SOURCE` or `__HAL_RCC_GET_USB_SOURCE` to get respectively RNG or UBS clock sources.

`__HAL_RCC_HSISTOP_ENABLE`

Description:

- Macros to enable or disable the force of the Internal High Speed oscillator (HSI) in STOP mode to be quickly available as kernel clock for USART and I2C.

Return value:

- None

Notes:

- The Enable of this function has not effect on the HSION bit.

`__HAL_RCC_HSISTOP_DISABLE`

`__HAL_RCC_LSEDRIVE_CONFIG`

Description:

- Macro to configures the External Low Speed oscillator (LSE) drive capability.

Parameters:

- `__RCC_LSEDrive__`: specifies the new state of the LSE drive capability. This parameter can be one of the following values:
 - `RCC_LSEDRIVE_LOW`: LSE oscillator low drive capability.
 - `RCC_LSEDRIVE_MEDIUMLOW`: LSE oscillator medium low drive capability.
 - `RCC_LSEDRIVE_MEDIUMHIGH`: LSE oscillator medium high drive capability.
 - `RCC_LSEDRIVE_HIGH`: LSE oscillator high drive capability.

Return value:

- None

`__HAL_RCC_WAKEUPSTOP_CLK_CONFIG`

Description:

- Macro to configures the wake up from

stop clock.

Parameters:

- `__RCC_STOPWUCLK__`: specifies the clock source used after wake up from stop. This parameter can be one of the following values:
 - `RCC_STOP_WAKEUPCLOCK_MSI`: MSI selected as system clock source
 - `RCC_STOP_WAKEUPCLOCK_HSI`: HSI selected as system clock source

Return value:

- None

Description:

- Enables the specified CRS interrupts.

Parameters:

- `__INTERERRUPT__`: 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

Description:

- Disables the specified CRS interrupts.

Parameters:

- `__INTERERRUPT__`: 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

Description:

- Check the CRS interrupt has occurred or not.

Parameters:

`__HAL_RCC_CRs_ENABLE_IT`

`__HAL_RCC_CRs_DISABLE_IT`

`__HAL_RCC_CRs_GET_IT_SOURCE`

RCC_CR_S_IT_ERROR_MASK

- `__INTERRUPT__`: specifies the CRS interrupt source to check. This parameter can be one of the following values:
 - `RCC_CR_S_IT_SYNCOK`
 - `RCC_CR_S_IT_SYNCWARN`
 - `RCC_CR_S_IT_ERR`
 - `RCC_CR_S_IT_ESYNC`

Return value:

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

Description:

- Clear the CRS interrupt pending 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_CR_S_IT_SYNCOK`
 - `RCC_CR_S_IT_SYNCWARN`
 - `RCC_CR_S_IT_ERR`
 - `RCC_CR_S_IT_ESYNC`
 - `RCC_CR_S_IT_TRIMOVF`
 - `RCC_CR_S_IT_SYNCERR`
 - `RCC_CR_S_IT_SYNCMISS`

`__HAL_RCC_CR_S_CLEAR_IT`

`__HAL_RCC_CR_S_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_CR_S_FLAG_SYNCOK`
 - `RCC_CR_S_FLAG_SYNCWARN`
 - `RCC_CR_S_FLAG_ERR`
 - `RCC_CR_S_FLAG_ESYNC`
 - `RCC_CR_S_FLAG_TRIMOVF`
 - `RCC_CR_S_FLAG_SYNCERR`
 - `RCC_CR_S_FLAG_SYNCMISS`

Return value:

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

RCC_CR_S_FLAG_ERROR_MASK

Description:

	• Clears the CRS specified FLAG. Parameters: • <code>_FLAG_</code>: specifies the flag to clear. This parameter can be one of the following values: – <code>RCC_CRs_FLAG_SYNCOK</code> – <code>RCC_CRs_FLAG_SYNCWARN</code> – <code>RCC_CRs_FLAG_ERR</code> – <code>RCC_CRs_FLAG_ESYNC</code> – <code>RCC_CRs_FLAG_TRIMOVF</code> – <code>RCC_CRs_FLAG_SYNCERR</code> – <code>RCC_CRs_FLAG_SYNCMISS</code> Return value: • None
<code>__HAL_RCC_CRs_CLEAR_FLAG</code> <code>__HAL_RCC_CRs_FREQ_ERROR_COUNT</code> <code>ER_ENABLE</code>	Description: • Enables the oscillator clock for frequency error counter. Return value: • None Notes: • when the CEN bit is set the <code>CRs_CFGR</code> register becomes write-protected.
<code>__HAL_RCC_CRs_FREQ_ERROR_COUNT</code> <code>ER_DISABLE</code>	Description: • Disables the oscillator clock for frequency error counter. Return value: • None
<code>__HAL_RCC_CRs_AUTOMATIC_CALIB_E</code> <code>NABLE</code>	Description: • Enables the automatic hardware adjustment of TRIM bits. Return value: • None Notes: • When the AUTOTRIMEN bit is set the <code>CRs_CFGR</code> register becomes write-protected.
<code>__HAL_RCC_CRs_AUTOMATIC_CALIB_DI</code> <code>SABLE</code>	Description: • Enables or disables the automatic hardware adjustment of TRIM bits.

`__HAL_RCC_CRs_RELOADVALUE_CALCULATE`

Return value:

- None

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: $RELOAD = (f_{TARGET} / f_{SYNC}) - 1$

`__HAL_RCC_HSI_OUT_ENABLE`

Notes:

- After reset, the HSI output is not available

`__HAL_RCC_HSI_OUT_DISABLE`

`__HAL_RCC_HSI48_ENABLE`

Notes:

- After enabling the HSI48, the application software should wait on HSI48RDY flag to be set indicating that HSI48 clock is stable and can be used to clock the USB. The HSI48 is stopped by hardware when entering STOP and STANDBY modes.

`__HAL_RCC_HSI48_DISABLE`

`__HAL_RCC_HSI48M_DIV6_OUT_ENABLE`

Notes:

- After reset, the HSI48Mhz (divided by 6) output is not available

`__HAL_RCC_HSI48M_DIV6_OUT_DISABLE`

APB2 Peripheral Force Release Reset

`__HAL_RCC_USART1_FORCE_RESET`

`__HAL_RCC_ADC1_FORCE_RESET`

__HAL_RCC_SPI1_FORCE_RESET
 __HAL_RCC_TIM21_FORCE_RESET
 __HAL_RCC_TIM22_FORCE_RESET
 __HAL_RCC_USART1_RELEASE_RESET
 __HAL_RCC_ADC1_RELEASE_RESET
 __HAL_RCC_SPI1_RELEASE_RESET
 __HAL_RCC_TIM21_RELEASE_RESET
 __HAL_RCC_TIM22_RELEASE_RESET

RCC CRS Error Limit Default

RCC_CRSErrorLIMIT_DEFAULT Default Frequency error limit

RCC CRS Flags

RCC_CRSError_FLAG_SYNCOK
 RCC_CRSError_FLAG_SYNCWARN
 RCC_CRSError_FLAG_ERR
 RCC_CRSError_FLAG_ESYNC
 RCC_CRSError_FLAG_TRIMOVF Trimming overflow or underflow
 RCC_CRSError_FLAG_SYNCERR SYNC error
 RCC_CRSError_FLAG_SYNCMISS SYNC missed

RCC CRS Frequency Error Direction

RCC_CRSError_FREQERRORDIR_UP Upcounting direction, the actual frequency is above the target
 RCC_CRSError_FREQERRORDIR_DOWN Downcounting direction, the actual frequency is below the target

RCC CRS HSI48 Calibration Default

RCC_CRSError_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

RCC CRS Interrupt Sources

RCC_CRSError_IT_SYNCOK SYNC event OK
 RCC_CRSError_IT_SYNCWARN SYNC warning
 RCC_CRSError_IT_ERR error
 RCC_CRSError_IT_ESYNC Expected SYNC
 RCC_CRSError_IT_TRIMOVF Trimming overflow or underflow
 RCC_CRSError_IT_SYNCERR SYNC error
 RCC_CRSError_IT_SYNCMISS SYNC missed

RCC CRS Reload Default Value

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).

RCC CRs Synchro Divider

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

RCC CRs Synchro Polarity

RCC_CRs_SYNC_POLARITY_RISING Synchro Active on rising edge (default)
 RCC_CRs_SYNC_POLARITY_FALLING Synchro Active on falling edge

RCC CRs Synchro Source

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)

RCCEX Exported Constants

RCC_CRs_NONE
 RCC_CRs_TIMEOUT
 RCC_CRs_SYNCOK
 RCC_CRs_SYNCWARM
 RCC_CRs_SYNCERR
 RCC_CRs_SYNCMISS
 RCC_CRs_TRIMOV

RCC LSE CSS external interrupt line

RCC_EXTI_LINE_LSECSS External interrupt line 19 connected to the LSE CSS EXTI Line

RCC HSI48M Clock Source

RCC_FLAG_HSI48
 RCC_HSI48M_PLL
 RCC_HSI48M_HSI48

RCC I2C1 Clock Source

RCC_I2C1CLKSOURCE_PCLK1
 RCC_I2C1CLKSOURCE_SYSCLK

RCC_I2C1CLKSOURCE_HSI

RCC I2C3 Clock Source

RCC_I2C3CLKSOURCE_PCLK1

RCC_I2C3CLKSOURCE_SYSCLK

RCC_I2C3CLKSOURCE_HSI

IOPORT Peripheral Clock Enable Disable

__HAL_RCC_GPIOE_CLK_ENABLE

__HAL_RCC_GPIOE_CLK_DISABLE

__HAL_RCC_GPIOD_CLK_ENABLE

__HAL_RCC_GPIOD_CLK_DISABLE

IOPORT Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPIOE_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOE_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOD_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOD_CLK_SLEEP_DISABLE

IOPORT Peripheral Force Release Reset

__HAL_RCC_GPIOE_FORCE_RESET

__HAL_RCC_GPIOE_RELEASE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

RCC LPTIM1 Clock Source

RCC_LPTIM1CLKSOURCE_PCLK

RCC_LPTIM1CLKSOURCE_LSI

RCC_LPTIM1CLKSOURCE_HSI

RCC_LPTIM1CLKSOURCE_LSE

RCC LPUART Clock Source

RCC_LPUART1CLKSOURCE_PCLK1

RCC_LPUART1CLKSOURCE_SYSCLK

RCC_LPUART1CLKSOURCE_HSI

RCC_LPUART1CLKSOURCE_LSE

RCC LSE Drive Configuration

RCC_LSEDRIVE_LOW

RCC_LSEDRIVE_MEDIUMLOW

RCC_LSEDRIVE_MEDIUMHIGH

RCC_LSEDRIVE_HIGH

AHB Peripheral Clock Enable Disable

__HAL_RCC_AES_CLK_ENABLE
__HAL_RCC_AES_CLK_DISABLE
__HAL_RCC_TSC_CLK_ENABLE
__HAL_RCC_TSC_CLK_DISABLE
__HAL_RCC_RNG_CLK_ENABLE
__HAL_RCC_RNG_CLK_DISABLE
__HAL_RCC_LSECSS_EXTI_ENABLE_IT

Description:

- Enable interrupt on RCC LSE CSS EXTI Line 19.

Return value:

- None

__HAL_RCC_LSECSS_EXTI_DISABLE_IT

Description:

- Disable interrupt on RCC LSE CSS EXTI Line 19.

Return value:

- None

__HAL_RCC_LSECSS_EXTI_ENABLE_EVENT

Description:

- Enable event on RCC LSE CSS EXTI Line 19.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_DISABLE_EVENT

Description:

- Disable event on RCC LSE CSS EXTI Line 19.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_ENABLE_FALLING_EDGE

Description:

- RCC LSE CSS EXTI line configuration: set falling edge trigger.

Return value:

`__HAL_RCC_LSECSS_EXTI_DISABLE_FALLING_EDGE`

- None.

Description:

- Disable the RCC LSE CSS Extended Interrupt Falling Trigger.

Return value:

- None.

`__HAL_RCC_LSECSS_EXTI_ENABLE_RISING_EDGE`

Description:

- RCC LSE CSS EXTI line configuration: set rising edge trigger.

Return value:

- None.

`__HAL_RCC_LSECSS_EXTI_DISABLE_RISING_EDGE`

Description:

- Disable the RCC LSE CSS Extended Interrupt Rising Trigger.

Return value:

- None.

`__HAL_RCC_LSECSS_EXTI_ENABLE_RISING_FALLING_EDGE`

Description:

- RCC LSE CSS EXTI line configuration: set rising & falling edge trigger.

Return value:

- None.

`__HAL_RCC_LSECSS_EXTI_DISABLE_RISING_FALLING_EDGE`

Description:

- Disable the RCC LSE CSS Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_GET_FLAG
Description:

- Check whether the specified RCC LSE CSS EXTI interrupt flag is set or not.

Return value:

- EXTI: RCC LSE CSS Line Status.

__HAL_RCC_LSECSS_EXTI_CLEAR_FLAG

Description:

- Clear the RCC LSE CSS EXTI flag.

Return value:

- None.

__HAL_RCC_LSECSS_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None.

RCC Periph Clock Selection

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_USART2

RCC_PERIPHCLK_LPUART1

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2C2

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_USB

RCC_PERIPHCLK_LPTIM1

RCC_PERIPHCLK_LCD

RCC_PERIPHCLK_I2C3

RCC RNG Clock Source

RCC_RNGCLKSOURCE_HSI48

RCC_RNGCLKSOURCE_PLLCLK

RCC StopWakeUp Clock

RCC_STOP_WAKEUPCLOCK_MSI

RCC_STOP_WAKEUPCLOCK_HSI

RCC TIM Prescaler Selection

RCC_TIMPRES_DESACTIVATED

RCC_TIMPRES_ACTIVATED

RCC USART1 Clock Source

RCC_USART1CLKSOURCE_PCLK2

RCC_USART1CLKSOURCE_SYSCLK

RCC_USART1CLKSOURCE_HSI

RCC_USART1CLKSOURCE_LSE

RCC USART2 Clock Source

RCC_USART2CLKSOURCE_PCLK1

RCC_USART2CLKSOURCE_SYSCLK

RCC_USART2CLKSOURCE_HSI

RCC_USART2CLKSOURCE_LSE

RCC USB Clock Source

RCC_USBCLKSOURCE_HSI48

RCC_USBCLKSOURCE_PLL

37 HAL RNG Generic Driver

37.1 RNG Firmware driver registers structures

37.1.1 RNG_HandleTypeDef

Data Fields

- *RNG_TypeDef * Instance*
- *HAL_LockTypeDef Lock*
- *__IO HAL_RNG_StateTypeDef State*
- *uint32_t RandomNumber*

Field Documentation

- *RNG_TypeDef* RNG_HandleTypeDef::Instance*
Register base address
- *HAL_LockTypeDef RNG_HandleTypeDef::Lock*
RNG locking object
- *__IO HAL_RNG_StateTypeDef RNG_HandleTypeDef::State*
RNG communication state
- *uint32_t RNG_HandleTypeDef::RandomNumber*
Last Generated RNG Data

37.2 RNG Firmware driver API description

37.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using `__HAL_RCC_RNG_CLK_ENABLE()` macro. in `HAL_RNG_MspInit()`.
2. Activate the RNG peripheral using `HAL_RNG_Init()` function.
3. Wait until the 32 bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using `HAL_RNG_GenerateRandomNumber()` function.

37.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the RNG according to the specified parameters in the `RNG_InitTypeDef` and create the associated handle
- DeInitialize the RNG peripheral
- Initialize the RNG MSP
- DeInitialize RNG MSP

This section contains the following APIs:

- [*HAL_RNG_Init\(\)*](#)
- [*HAL_RNG_DeInit\(\)*](#)

- [HAL_RNG_MspInit\(\)](#)
- [HAL_RNG_MspDeInit\(\)](#)

37.2.3 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- Handle RNG interrupt request

This section contains the following APIs:

- [HAL_RNG_GenerateRandomNumber\(\)](#)
- [HAL_RNG_GenerateRandomNumber_IT\(\)](#)
- [HAL_RNG_IRQHandler\(\)](#)
- [HAL_RNG_GetRandomNumber\(\)](#)
- [HAL_RNG_GetRandomNumber_IT\(\)](#)
- [HAL_RNG_ReadLastRandomNumber\(\)](#)
- [HAL_RNG_ReadyDataCallback\(\)](#)
- [HAL_RNG_ErrorCallback\(\)](#)

37.2.4 Peripheral State functions

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

This section contains the following APIs:

- [HAL_RNG_GetState\(\)](#)

37.2.5 HAL_RNG_Init

Function Name	HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)
Function Description	Initializes the RNG peripheral and creates the associated handle.
Parameters	• hrng: pointer to a RNG_HandleTypeDef structure.
Return values	• HAL: status

37.2.6 HAL_RNG_DeInit

Function Name	HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)
Function Description	DeInitializes the RNG peripheral.
Parameters	• hrng: pointer to a RNG_HandleTypeDef structure.
Return values	• HAL: status

37.2.7 HAL_RNG_MspInit

Function Name	void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)
Function Description	Initializes the RNG MSP.
Parameters	• hrng: pointer to a RNG_HandleTypeDef structure.
Return values	• None:

37.2.8 HAL_RNG_MspDeInit

Function Name	void HAL_RNG_MspDeInit (RNG_HandleTypeDef * hrng)
Function Description	DeInitializes the RNG MSP.
Parameters	hrng: pointer to a RNG_HandleTypeDef structure.
Return values	None:

37.2.9 HAL_RNG_GenerateRandomNumber

Function Name	HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random32bit)
Function Description	Generates a 32-bit random number.
Parameters	hrng: pointer to a RNG_HandleTypeDef structure. random32bit: pointer to generated random number variable if successful.
Return values	HAL: status
Notes	Each time the random number data is read the RNG_FLAG_DRDY flag is automatically cleared.

37.2.10 HAL_RNG_GenerateRandomNumber_IT

Function Name	HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber_IT (RNG_HandleTypeDef * hrng)
Function Description	Generates a 32-bit random number in interrupt mode.
Parameters	hrng: pointer to a RNG_HandleTypeDef structure.
Return values	HAL: status

37.2.11 HAL_RNG_IRQHandler

Function Name	void HAL_RNG_IRQHandler (RNG_HandleTypeDef * hrng)
Function Description	Handles RNG interrupt request.
Parameters	hrng: pointer to a RNG_HandleTypeDef structure.
Return values	None:
Notes	- In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using <code>__HAL_RNG_CLEAR_IT()</code>. The clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can be used. - In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using <code>__HAL_RNG_CLEAR_IT()</code>, then disable and enable the RNG peripheral to reinitialize and restart the RNG.

- User-written HAL_RNG_ErrorCallback() API is called once whether SEIS or CEIS are set.

37.2.12 HAL_RNG_GetRandomNumber

Function Name	uint32_t HAL_RNG_GetRandomNumber (RNG_HandleTypeDef * hrng)
Function Description	return generated random number in polling mode (Obsolete).
Parameters	• hrng: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	• random: value

37.2.13 HAL_RNG_GetRandomNumber_IT

Function Name	uint32_t HAL_RNG_GetRandomNumber_IT (RNG_HandleTypeDef * hrng)
Function Description	Returns a 32-bit random number with interrupt enabled (Obsolete), Use HAL_RNG_GenerateRandomNumber_IT() API instead.
Parameters	• hrng: RNG handle
Return values	• 32-bit: random number

37.2.14 HAL_RNG_ReadLastRandomNumber

Function Name	uint32_t HAL_RNG_ReadLastRandomNumber (RNG_HandleTypeDef * hrng)
Function Description	Read latest generated random number.
Parameters	• hrng: pointer to a RNG_HandleTypeDef structure.
Return values	• random: value

37.2.15 HAL_RNG_ReadyDataCallback

Function Name	void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)
Function Description	Data Ready callback in non-blocking mode.
Parameters	• hrng: pointer to a RNG_HandleTypeDef structure.. • random32bit: generated random value
Return values	• None:

37.2.16 HAL_RNG_ErrorCallback

Function Name	void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * hrng)
Function Description	RNG error callbacks.
Parameters	• hrng: pointer to a RNG_HandleTypeDef structure.
Return values	• None:

37.2.17 HAL_RNG_GetState



Function Name	HAL_RNG_StateTypeDef HAL_RNG_GetState (RNG_HandleTypeDef * hrng)
Function Description	Returns the RNG state.
Parameters	hrng: pointer to a RNG_HandleTypeDef structure.
Return values	HAL: state

37.3 RNG Firmware driver defines

37.3.1 RNG

RNG Interrupt definition

RNG_IT_DRDY	Data ready interrupt
RNG_IT_CEI	Clock error interrupt
RNG_IT_SEI	Seed error interrupt

RNG Flag definition

RNG_FLAG_DRDY	Data ready
RNG_FLAG_CECS	Clock error current status
RNG_FLAG_SECS	Seed error current status

RNG Exported Macros

__HAL_RNG_RESET_HANDLE_STATE	Description: - Reset RNG handle state. Parameters: __HANDLE__: RNG Handle Return value: - None
__HAL_RNG_ENABLE	Description: - Enables the RNG peripheral. Parameters: __HANDLE__: RNG Handle Return value: - None
__HAL_RNG_DISABLE	Description: - Disables the RNG peripheral. Parameters: __HANDLE__: RNG Handle Return value: - None
__HAL_RNG_GET_FLAG	Description:

- Check the selected RNG flag status.

Parameters:

- `__HANDLE__`: RNG Handle
- `__FLAG__`: RNG flag This parameter can be one of the following values:
 - `RNG_FLAG_DRDY`: Data ready
 - `RNG_FLAG_CECS`: Clock error current status
 - `RNG_FLAG_SECS`: Seed error current status

Return value:

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

Description:

- Clears the selected RNG flag status.

Parameters:

- `__HANDLE__`: RNG handle
- `__FLAG__`: RNG flag to clear

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment, flags `RNG_FLAG_DRDY`, `RNG_FLAG_CECS` and `RNG_FLAG_SECS` are read-only.

Description:

- Enables the RNG interrupts.

Parameters:

- `__HANDLE__`: RNG Handle

Return value:

- None

Description:

- Disables the RNG interrupts.

Parameters:

- `__HANDLE__`: RNG Handle

Return value:

- None

Description:

- Checks whether the specified RNG interrupt has occurred or not.

`__HAL_RNG_CLEAR_FLAG``__HAL_RNG_ENABLE_IT``__HAL_RNG_DISABLE_IT``__HAL_RNG_GET_IT`

__HAL_RNG_CLEAR_IT**Parameters:**

- `__HANDLE__`: RNG Handle
- `__INTERRUPT__`: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
 - `RNG_IT_DRDY`: Data ready interrupt
 - `RNG_IT_CEI`: Clock error interrupt
 - `RNG_IT_SEI`: Seed error interrupt

Return value:

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

Description:

- Clears the RNG interrupt status flags.

Parameters:

- `__HANDLE__`: RNG Handle
- `__INTERRUPT__`: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
 - `RNG_IT_CEI`: Clock error interrupt
 - `RNG_IT_SEI`: Seed error interrupt

Return value:

- None

Notes:

- `RNG_IT_DRDY` flag is read-only, reading `RNG_DR` register automatically clears `RNG_IT_DRDY`.

38 HAL RTC Generic Driver

38.1 RTC Firmware driver registers structures

38.1.1 RTC_InitTypeDef

Data Fields

- *uint32_t HourFormat*
- *uint32_t AsynchPrediv*
- *uint32_t SynchPrediv*
- *uint32_t OutPut*
- *uint32_t OutPutRemap*
- *uint32_t OutPutPolarity*
- *uint32_t OutPutType*

Field Documentation

- *uint32_t RTC_InitTypeDef::HourFormat*
Specifies the RTC Hour Format. This parameter can be a value of [RTC_Hour_Formats](#)
- *uint32_t RTC_InitTypeDef::AsynchPrediv*
Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F
- *uint32_t RTC_InitTypeDef::SynchPrediv*
Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFF
- *uint32_t RTC_InitTypeDef::OutPut*
Specifies which signal will be routed to the RTC output. This parameter can be a value of [RTCEx_Output_selection_Definitions](#)
- *uint32_t RTC_InitTypeDef::OutPutRemap*
Specifies the remap for RTC output. This parameter can be a value of [RTC_Output_ALARM_OUT_Remap](#)
- *uint32_t RTC_InitTypeDef::OutPutPolarity*
Specifies the polarity of the output signal. This parameter can be a value of [RTC_Output_Polarity_Definitions](#)
- *uint32_t RTC_InitTypeDef::OutPutType*
Specifies the RTC Output Pin mode. This parameter can be a value of [RTC_Output_Type_ALARM_OUT](#)

38.1.2 RTC_TimeTypeDef

Data Fields

- *uint8_t Hours*
- *uint8_t Minutes*
- *uint8_t Seconds*
- *uint8_t TimeFormat*

- ***uint32_t SubSeconds***
- ***uint32_t SecondFraction***
- ***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
- ***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::SubSeconds***
Specifies the RTC_SSR RTC Sub Second register content. This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity
- ***uint32_t RTC_TimeTypeDef::SecondFraction***
Specifies the range or granularity of Sub Second register content corresponding to Synchronous pre-scaler factor value (PREDIV_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL_RTC_GetTime function
- ***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](#)

38.1.3 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

38.1.4 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](#)

38.1.5 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

38.2 RTC Firmware driver API description

38.2.1 Backup Domain Operating Condition

As long as the supply voltage remains in the operating range, the RTC never stops, regardless of the device status (Run mode, low power modes or under reset).

38.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_CSR register to their reset values.

A backup domain reset is generated when one of the following events occurs:

- Software reset, triggered by setting the RTCRST bit in the RCC Control Status register (RCC_CSR).
- Power reset (BOR/POR/PDR).

38.2.3 Backup Domain Access

After reset, the backup domain (RTC registers and RTC backup data registers) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the `__HAL_RCC_PWR_CLK_ENABLE()` function.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.
- Select the RTC clock source using the `__HAL_RCC_RTC_CONFIG()` function.
- Enable RTC Clock using the `__HAL_RCC_RTC_ENABLE()` function.

38.2.4 How to use RTC Driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the `HAL_RTC_Init()` function.

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
- To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.

Alarm configuration

- To configure the RTC Alarm use the HAL_RTC_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL_RTC_SetAlarm_IT() function.
- To read the RTC Alarm, use the HAL_RTC_GetAlarm() function.

38.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

38.2.6 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
 - A 7-bit asynchronous prescaler and a 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 wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers. The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- [*HAL_RTC_Init\(\)*](#)
- [*HAL_RTC_DeInit\(\)*](#)
- [*HAL_RTC_MsPlnit\(\)*](#)

- [HAL_RTC_MspDeInit\(\)](#)

38.2.7 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\(\)](#)

38.2.8 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\(\)](#)

38.2.9 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\(\)](#)

38.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- [HAL_RTC_GetState\(\)](#)

38.2.11 HAL_RTC_Init

Function Name	HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)
Function Description	Initialize the RTC peripheral.
Parameters	• hrtc: RTC handle
Return values	• HAL: status

38.2.12 HAL_RTC_DeInit

Function Name	HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)

Function Description	DeInitialize the RTC peripheral.
Parameters	• hrtc: RTC handle
Return values	• HAL: status
Notes	• This function doesn't reset the RTC Backup Data registers.

38.2.13 HAL_RTC_MspInit

Function Name	void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)
Function Description	Initialize the RTC MSP.
Parameters	• hrtc: RTC handle
Return values	• None:

38.2.14 HAL_RTC_MspDeInit

Function Name	void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)
Function Description	DeInitialize the RTC MSP.
Parameters	• hrtc: RTC handle
Return values	• None:

38.2.15 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	• 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

38.2.16 HAL_RTC_GetTime

Function Name	HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
Function Description	Get RTC current time.
Parameters	• hrtc: RTC handle • sTime: Pointer to Time structure with Hours, Minutes and Seconds fields returned with input format (BIN or BCD), also SubSeconds field returning the RTC_SSR register content and SecondFraction field the Synchronous pre-scaler factor to be used for second fraction ratio computation. • 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

Notes

- You can use SubSeconds and SecondFraction (sTime structure fields returned) to convert SubSeconds value in second fraction ratio with time unit following generic formula: Second fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S>= SS
- 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.

38.2.17 HAL_RTC_SetDate

Function Name

HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function Description

Set RTC current date.

Parameters

- **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

- **HAL:** status

38.2.18 HAL_RTC_GetDate

Function Name

HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function Description

Get RTC current date.

Parameters

- **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

- **HAL:** status

Notes

- 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.

38.2.19 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	• 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	• HAL: status

38.2.20 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	• 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	• HAL: status
Notes	• 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.

38.2.21 HAL_RTC_DeactivateAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)
Function Description	Deactivate the specified RTC Alarm.
Parameters	• hrtc: RTC handle • Alarm: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmA AlarmARTC_ALARM_B: AlarmB
Return values	• HAL: status

38.2.22 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	• 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, RTC_ALARM_B: AlarmB • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data format, RTC_FORMAT_BCD: BCD data format
Return values	• HAL: status

38.2.23 HAL_RTC_AlarmIRQHandler

Function Name	void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	Handle Alarm interrupt request.
Parameters	• hrtc: RTC handle
Return values	• None:

38.2.24 HAL_RTC_AlarmAEventCallback

Function Name	void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Alarm A callback.
Parameters	• hrtc: RTC handle
Return values	• None:

38.2.25 HAL_RTC_PollForAlarmAEvent

Function Name	HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	Handle AlarmA Polling request.
Parameters	• hrtc: RTC handle • Timeout: Timeout duration
Return values	• HAL: status

38.2.26 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	• hrtc: RTC handle
Return values	• HAL: status
Notes	• 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.

38.2.27 HAL_RTC_GetState

Function Name	HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)
Function Description	Return the RTC handle state.
Parameters	hrtc: RTC handle
Return values	HAL: state

38.3 RTC Firmware driver defines

38.3.1 RTC

RTC AlarmDateWeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE
RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

RTC AlarmMask 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_B

RTC Alarm Sub Seconds Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL	All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm
RTC_ALARMSUBSECONDMASK_SS14_1	SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.
RTC_ALARMSUBSECONDMASK_SS14_2	SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_3	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 DayLightSaving 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.

`__HAL_RTC_WRITEPROTECTION_ENABLE`

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Enable the write protection for RTC registers.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Enable the RTC ALARMA peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Disable the RTC ALARMA peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Enable the RTC ALARMB peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Disable the RTC ALARMB

`__HAL_RTC_ALARMA_ENABLE`

`__HAL_RTC_ALARMA_DISABLE`

`__HAL_RTC_ALARMB_ENABLE`

`__HAL_RTC_ALARMB_DISABLE`

peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

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
 - `RTC_IT_ALRB`: Alarm B interrupt

Return value:

- None

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
 - `RTC_IT_ALRA`: Alarm A interrupt
 - `RTC_IT_ALRB`: Alarm B interrupt

Return value:

- None

Description:

`__HAL_RTC_ALARM_ENABLE_IT`

`__HAL_RTC_ALARM_DISABLE_IT`

`__HAL_RTC_ALARM_GET_IT`

`__HAL_RTC_ALARM_GET_IT_SOURCE`

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Alarm interrupt sources to check. This parameter can be:
 - `RTC_IT_ALRA`: Alarm A interrupt
 - `RTC_IT_ALRB`: Alarm B interrupt

Return value:

- None

Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Alarm interrupt sources to check. This parameter can be:
 - `RTC_IT_ALRA`: Alarm A interrupt
 - `RTC_IT_ALRB`: Alarm B interrupt

Return value:

- None

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_ALRBF`

- RTC_FLAG_ALRAWF
- RTC_FLAG_ALRBWF

Return value:

- None

Description:

- Clear the RTC Alarm's pending flags.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag sources to clear. This parameter can be:
 - RTC_FLAG_ALRAF
 - RTC_FLAG_ALRBF

Return value:

- None

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.

`__HAL_RTC_ALARM_CLEAR_FLAG`

`__HAL_RTC_ALARM_EXTI_ENABLE_IT`

`__HAL_RTC_ALARM_EXTI_DISABLE_IT`

`__HAL_RTC_ALARM_EXTI_ENABLE_EVENT`

`__HAL_RTC_ALARM_EXTI_DISABLE_EVENT`

<code>__HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE</code>	Description: • Enable falling edge trigger on the RTC Alarm associated Exti line. Return value: • None.
<code>__HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE</code>	Description: • Disable falling edge trigger on the RTC Alarm associated Exti line. Return value: • None.
<code>__HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE</code>	Description: • Enable rising edge trigger on the RTC Alarm associated Exti line. Return value: • None.
<code>__HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE</code>	Description: • Disable rising edge trigger on the RTC Alarm associated Exti line. Return value: • None.
<code>__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE</code>	Description: • Enable rising & falling edge trigger on the RTC Alarm associated Exti line. Return value: • None.
<code>__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE</code>	Description: • Disable rising & falling edge trigger on the RTC Alarm associated Exti line. Return value: • None.
<code>__HAL_RTC_ALARM_EXTI_GET_FLAG</code>	Description: • Check whether the RTC Alarm associated Exti line interrupt flag is set or not. Return value:

__HAL_RTC_ALARM_EXTI_CLEAR_FLAG

- Line: Status.

Description:

- Clear the RTC Alarm associated Exti line flag.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on RTC Alarm associated Exti line.

Return value:

- None.

RTC Flags Definitions

RTC_FLAG_RECALPF

RTC_FLAG_TAMP2F

RTC_FLAG_TAMP1F

RTC_FLAG_TSOVF

RTC_FLAG_TSF

RTC_FLAG_WUTF

RTC_FLAG_ALRBF

RTC_FLAG_ALRAF

RTC_FLAG_INITF

RTC_FLAG_RSF

RTC_FLAG_INITS

RTC_FLAG_SHPF

RTC_FLAG_WUTWF

RTC_FLAG_ALRBWF

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_ALRB

RTC_IT_ALRA

RTC_IT_TAMP

RTC_IT_TAMP1

RTC_IT_TAMP2

RTC Private macros to check input parameters

IS_RTC_HOUR_FORMAT

IS_RTC_OUTPUT_POL

IS_RTC_OUTPUT_TYPE

IS_RTC_OUTPUT_REMAP

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

IS_RTC_ASYNC_PREDIV

IS_RTC_SYNC_PREDIV

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

RTC Month Date Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

RTC Output ALARM OUT Remap

RTC_OUTPUT_REMAP_NONE

RTC_OUTPUT_REMAP_POS1

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_OPENDRAIN

RTC_OUTPUT_TYPE_PUSHPULL

RTC StoreOperation 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

39 HAL RTC Extension Driver

39.1 RTCEX Firmware driver registers structures

39.1.1 RTC_TamperTypeDef

Data Fields

- *uint32_t Tamper*
- *uint32_t Interrupt*
- *uint32_t Trigger*
- *uint32_t NoErase*
- *uint32_t MaskFlag*
- *uint32_t Filter*
- *uint32_t SamplingFrequency*
- *uint32_t PrechargeDuration*
- *uint32_t TamperPullUp*
- *uint32_t TimeStampOnTamperDetection*

Field Documentation

- *uint32_t RTC_TamperTypeDef::Tamper*
Specifies the Tamper Pin. This parameter can be a value of [RTCEX_Tamper_Pins_Definitions](#)
- *uint32_t RTC_TamperTypeDef::Interrupt*
Specifies the Tamper Interrupt. This parameter can be a value of [RTCEX_Tamper_Interrupt_Definitions](#)
- *uint32_t RTC_TamperTypeDef::Trigger*
Specifies the Tamper Trigger. This parameter can be a value of [RTCEX_Tamper_Trigger_Definitions](#)
- *uint32_t RTC_TamperTypeDef::NoErase*
Specifies the Tamper no erase mode. This parameter can be a value of [RTCEX_Tamper_EraseBackUp_Definitions](#)
- *uint32_t RTC_TamperTypeDef::MaskFlag*
Specifies the Tamper Flag masking. This parameter can be a value of [RTCEX_Tamper_MaskFlag_Definitions](#)
- *uint32_t RTC_TamperTypeDef::Filter*
Specifies the RTC Filter Tamper. This parameter can be a value of [RTCEX_Tamper_Filter_Definitions](#)
- *uint32_t RTC_TamperTypeDef::SamplingFrequency*
Specifies the sampling frequency. This parameter can be a value of [RTCEX_Tamper_Sampling_Frequencies_Definitions](#)
- *uint32_t RTC_TamperTypeDef::PrechargeDuration*
Specifies the Precharge Duration . This parameter can be a value of [RTCEX_Tamper_Pin_Precharge_Duration_Definitions](#)
- *uint32_t RTC_TamperTypeDef::TamperPullUp*
Specifies the Tamper PullUp . This parameter can be a value of [RTCEX_Tamper_Pull_UP_Definitions](#)

- ***uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection***
Specifies the TimeStampOnTamperDetection. This parameter can be a value of [*RTCEx_Tamper_TimeStampOnTamperDetection_Definitions*](#)

39.2 RTCEx Firmware driver API description

39.2.1 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\(\)*](#)

39.2.2 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\(\)*](#)

39.2.3 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\(\)](#)

39.2.4 Extended features functions

This section provides functions allowing to:

- RTC Alarm B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- [HAL_RTCEX_AlarmBEventCallback\(\)](#)
- [HAL_RTCEX_PollForAlarmBEvent\(\)](#)

39.2.5 HAL_RTCEX_SetTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEX_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)
Function Description	Set TimeStamp.
Parameters	• 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_TIMESTAMPPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin on STM32L05x/6x/7x/8x and PA2 on STM32L03x/4x/2x/1x.
Return values	• HAL: status
Notes	• This API must be called before enabling the TimeStamp feature.

39.2.6 HAL_RTCEX_SetTimeStamp_IT

Function Name	HAL_StatusTypeDef HAL_RTCEX_SetTimeStamp_IT

(RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function Description	Set TimeStamp with Interrupt.
Parameters	• 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 on STM32L05x/6x/7x/8x and PA2 on STM32L03x/4x/2x/1x.
Return values	• HAL: status
Notes	• This API must be called before enabling the TimeStamp feature.

39.2.7 HAL_RTCEX_DeactivateTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEX_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)
Function Description	Deactivate TimeStamp.
Parameters	• hrtc: RTC handle
Return values	• HAL: status

39.2.8 HAL_RTCEX_GetTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEX_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)
Function Description	Get the RTC TimeStamp value.
Parameters	• 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	• HAL: status

39.2.9 HAL_RTCEX_SetTamper

Function Name	HAL_StatusTypeDef HAL_RTCEX_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Set Tamper.

Parameters	• hrtc: RTC handle • sTamper: Pointer to Tamper Structure.
Return values	• HAL: status
Notes	• By calling this API we disable the tamper interrupt for all tampers.

39.2.10 HAL_RTCEX_SetTamper_IT

Function Name	HAL_StatusTypeDef HAL_RTCEX_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Set Tamper with interrupt.
Parameters	• hrtc: RTC handle • sTamper: Pointer to RTC Tamper.
Return values	• HAL: status
Notes	• By calling this API we force the tamper interrupt for all tampers.

39.2.11 HAL_RTCEX_DeactivateTamper

Function Name	HAL_StatusTypeDef HAL_RTCEX_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)
Function Description	Deactivate Tamper.
Parameters	• hrtc: RTC handle • Tamper: Selected tamper pin. This parameter can be RTC_Tamper_1 and/or RTC_TAMPER_2 for STM32L05x/6x. This parameter can be any combination of RTC_TAMPER_1, RTC_TAMPER_2 and RTC_TAMPER_3 for STM32L01x/2x/3x/7x/8x.
Return values	• HAL: status

39.2.12 HAL_RTCEX_TamperTimeStampIRQHandler

Function Name	void HAL_RTCEX_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	Handle TimeStamp interrupt request.
Parameters	• hrtc: RTC handle
Return values	• None:

39.2.13 HAL_RTCEX_TimeStampEventCallback

Function Name	void HAL_RTCEX_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	TimeStamp callback.
Parameters	• hrtc: RTC handle
Return values	• None:

39.2.14 HAL_RTCEx_Tamper1EventCallback

Function Name	void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 1 callback.
Parameters	• hrtc: RTC handle
Return values	• None:

39.2.15 HAL_RTCEx_Tamper2EventCallback

Function Name	void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 2 callback.
Parameters	• hrtc: RTC handle
Return values	• None:

39.2.16 HAL_RTCEx_Tamper3EventCallback

Function Name	void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 3 callback.
Parameters	• hrtc: RTC handle
Return values	• None:

39.2.17 HAL_RTCEx_PollForTimeStampEvent

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	Handle TimeStamp polling request.
Parameters	• hrtc: RTC handle • Timeout: Timeout duration
Return values	• HAL: status

39.2.18 HAL_RTCEx_PollForTamper1Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	Handle Tamper 1 Polling.
Parameters	• hrtc: RTC handle • Timeout: Timeout duration
Return values	• HAL: status

39.2.19 HAL_RTCEx_PollForTamper2Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function Description	Handle Tamper 2 Polling.
Parameters	• hrtc: RTC handle • Timeout: Timeout duration
Return values	• HAL: status

39.2.20 HAL_RTCEx_PollForTamper3Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	Handle Tamper 3 Polling.
Parameters	• hrtc: RTC handle • Timeout: Timeout duration
Return values	• HAL: status

39.2.21 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	• hrtc: RTC handle • WakeUpCounter: Wake up counter • WakeUpClock: Wake up clock
Return values	• HAL: status

39.2.22 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	• hrtc: RTC handle • WakeUpCounter: Wake up counter • WakeUpClock: Wake up clock
Return values	• HAL: status

39.2.23 HAL_RTCEx_DeactivateWakeUpTimer

Function Name	uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)
Function Description	Deactivate wake up timer counter.
Parameters	• hrtc: RTC handle
Return values	• HAL: status

39.2.24 HAL_RTCEx_GetWakeUpTimer

	Function Name	uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)
	Function Description	Get wake up timer counter.
	Parameters	• hrtc: RTC handle
	Return values	• Counter: value
39.2.25	HAL_RTCEx_WakeUpTimerIRQHandler	
	Function Name	void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)
	Function Description	Handle Wake Up Timer interrupt request.
	Parameters	• hrtc: RTC handle
	Return values	• None:
39.2.26	HAL_RTCEx_WakeUpTimerEventCallback	
	Function Name	void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)
	Function Description	Wake Up Timer callback.
	Parameters	• hrtc: RTC handle
	Return values	• None:
39.2.27	HAL_RTCEx_PollForWakeUpTimerEvent	
	Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
	Function Description	Handle Wake Up Timer Polling.
	Parameters	• hrtc: RTC handle • Timeout: Timeout duration
	Return values	• HAL: status
39.2.28	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	• hrtc: RTC handle • BackupRegister: RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register. • Data: Data to be written in the specified RTC Backup data register.
	Return values	• None:
39.2.29	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	• hrtc: RTC handle • BackupRegister: RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.
Return values	• Read: value

39.2.30 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	• hrtc: RTC handle • SmoothCalibPeriod: Select the Smooth Calibration Period. This parameter 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 pulses. RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added. • SmoothCalibMinusPulsesValue: Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.
Return values	• HAL: status
Notes	• To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

39.2.31 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	• 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

- **HAL:** status

Notes

- When REFCKON is set, firmware must not write to Shift control register.

39.2.32 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

- **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

- **HAL:** status

39.2.33 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

- **hrtc:** RTC handle

Return values

- **HAL:** status

39.2.34 HAL_RTCEX_SetRefClock

Function Name HAL_StatusTypeDef HAL_RTCEX_SetRefClock (RTC_HandleTypeDef * hrtc)

Function Description Enable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

39.2.35 HAL_RTCEX_DeactivateRefClock

Function Name HAL_StatusTypeDef HAL_RTCEX_DeactivateRefClock (RTC_HandleTypeDef * hrtc)

Function Description Disable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

39.2.36 HAL_RTCEx_EnableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)
Function Description	Enable the Bypass Shadow feature.
Parameters	hrtc: RTC handle
Return values	HAL: status
Notes	When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

39.2.37 HAL_RTCEx_DisableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)
Function Description	Disable the Bypass Shadow feature.
Parameters	hrtc: RTC handle
Return values	HAL: status
Notes	When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

39.2.38 HAL_RTCEx_AlarmBEventCallback

Function Name	void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Alarm B callback.
Parameters	hrtc: RTC handle
Return values	None:

39.2.39 HAL_RTCEx_PollForAlarmBEvent

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	Handle Alarm B Polling request.
Parameters	hrtc: RTC handle Timeout: Timeout duration
Return values	HAL: status

39.3 RTCEx Firmware driver defines**39.3.1 RTCEx*****RTCEx Add 1 Second Parameter Definitions***

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`

RTCEx Flags Definitions

`RTC_FLAG_TAMP3F`

RTCEx Interrupts Definitions

`RTC_IT_TAMP3`

Private macros to check input parameters

`IS_RTC_OUTPUT`

`IS_RTC_BKP`

`IS_TIMESTAMP_EDGE`

`IS_RTC_TAMPER`

`IS_RTC_TAMPER_INTERRUPT`

`IS_RTC_TIMESTAMP_PIN`

`IS_RTC_TAMPER_TRIGGER`

`IS_RTC_TAMPER_ERASE_MODE`

`IS_RTC_TAMPER_MASKFLAG_STATE`

`IS_RTC_TAMPER_FILTER`

`IS_RTC_TAMPER_SAMPLING_FREQ`

`IS_RTC_TAMPER_PRECHARGE_DURATION`

`IS_RTC_TAMPER_TIMESTAMPONTAMPER_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_SHIFT_ADD1S`

`IS_RTC_CALIB_OUTPUT`

RTCEx Output Selection Definition

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTC_OUTPUT_ALARM_B

RTC_OUTPUT_WAKEUP

RTCEx Smooth calib Minus pulses Definitions

IS_RTC_SMOOTH_CALIB_MINUS

RTCEx Smooth calib period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_16SEC If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_8SEC If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK pulses

RTCEx Smooth calib Plus pulses Definitions

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 subbstited during a 32-second window = CALM[8:0]

RTCEx Subtract Fraction Of Second Value

IS_RTC_SHIFT_SUBFS

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:

__HAL_RTC_TAMPER2_ENABLE

- None

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: All tamper interrupts
- RTC_IT_TAMP1: Tamper1 interrupt
- RTC_IT_TAMP2: Tamper2 interrupt
- RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

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: All tamper interrupts
 - RTC_IT_TAMP1: Tamper1 interrupt
 - RTC_IT_TAMP2: Tamper2 interrupt
 - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

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:

`__HAL_RTC_TAMPER_DISABLE_IT`

`__HAL_RTC_TAMPER_GET_IT`

`__HAL_RTC_TAMPER_GET_IT_SOURCE`

- None

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`: All tampers interrupts
 - `RTC_IT_TAMP1`: Tamper1 interrupt
 - `RTC_IT_TAMP2`: Tamper2 interrupt
 - `RTC_IT_TAMP3`: Tamper3 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`: Tamper1 flag
 - `RTC_FLAG_TAMP2F`: Tamper2 flag
 - `RTC_FLAG_TAMP3F`: Tamper3 flag

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`: Tamper1

- flag
- RTC_FLAG_TAMP2F: Tamper2 flag
- RTC_FLAG_TAMP3F: Tamper3 flag

Return value:

- None

RTCEx Tamper EraseBackUp Definitions

RTC_TAMPER_ERASE_BACKUP_ENABLE

RTC_TAMPER_ERASE_BACKUP_DISABLE

RTCEx Tamper Filter Definitions

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 Interrupt Definitions

RTC_TAMPER1_INTERRUPT

RTC_TAMPER2_INTERRUPT

RTC_TAMPER3_INTERRUPT

RTC_ALL_TAMPER_INTERRUPT

RTCEx Tamper MaskFlag Definitions

RTC_TAMPERMASK_FLAG_DISABLE

RTC_TAMPERMASK_FLAG_ENABLE

RTCEx Tamper Pins Definition

RTC_TAMPER_1

RTC_TAMPER_2

RTC_TAMPER_3

RTCEx Tamper Pin Precharge Duration Definitions

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 Definitions

RTC_TAMPER_PULLUP_ENABLE Tamper pins are pre-charged before sampling

RTC_TAMPER_PULLUP_DISABLE Tamper pins pre-charge is disabled

RTCEx Tamper Sampling Frequencies Definitions

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768 Each of the tamper inputs are sampled with a frequency = $\text{RTCCLK} / 32768$

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384 Each of the tamper inputs are sampled with a frequency = $\text{RTCCLK} / 16384$

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192 Each of the tamper inputs are sampled with a frequency = $\text{RTCCLK} / 8192$

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096 Each of the tamper inputs are sampled with a frequency = $\text{RTCCLK} / 4096$

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048 Each of the tamper inputs are sampled with a frequency = $\text{RTCCLK} / 2048$

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024 Each of the tamper inputs are sampled with a frequency = $\text{RTCCLK} / 1024$

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512 Each of the tamper inputs are sampled with a frequency = $\text{RTCCLK} / 512$

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256 Each of the tamper inputs are sampled with a frequency = $\text{RTCCLK} / 256$

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:

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_IT`

- None

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:

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_FALLING_EDGE`

- None.

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.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_EDGE`

Description
:

- Enable rising edge trigger on the RTC Tamper and Timestamp associated

ted Exti
line.

**Return
value:**

- None.

**Description
:**

- Disable
rising
edge
trigger
on the
RTC
Tamper
and
Timesta
mp
associa
ted Exti
line.

**Return
value:**

- None.

**Description
:**

- Enable
rising &
falling
edge
trigger
on the
RTC
Tamper
and
Timesta
mp
associa
ted Exti
line.

**Return
value:**

- None.

**Description
:**

- Disable
rising &
falling
edge
trigger

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_EDGE

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_FALLING_
EDGE

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_FALLING_
EDGE

on the
RTC
Tamper
and
Timesta
mp
associa
ted Exti
line.

**Return
value:**

- None.

**Description
:**

- Check whether the RTC Tamper and Timestamp associated EXTI line interrupt flag is set or not.

**Return
value:**

- Line:
Status.

**Description
:**

- Clear the RTC Tamper and Timestamp associated EXTI line flag.

**Return
value:**

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GET_FLAG`

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_CLEAR_FLAG`

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GENERATE_SWIT
Description
:

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

RTCEx Tamper TimeStampOnTamperDetection Definitions

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE TimeStamp on Tamper Detection event saved

RTC_TIMESTAMPONTAMPERDETECTION_DISABLE TimeStamp on Tamper Detection event is not saved

RTCEx Tamper Trigger Definitions

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

__HAL_RTC_TIMESTAMP_ENABLE_IT

handle.

Return value:

- None

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_TIMESTAMP_PIN_DEFAULT*****RTCEx Time Stamp Edges definition*****RTC_TIMESTAMP_EDGE_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:

`__HAL_RTC_WAKEUPTIMER_GET_IT`

- `__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

`__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE`**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

`__HAL_RTC_WAKEUPTIMER_GET_FLAG`**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

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

Description:

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

Description:`__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG``__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_IT``__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_IT`

	• Disable interrupt on the RTC WakeUp Timer associated Exti line. Return value: • None
<code>__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_EVENT</code>	Description: • Enable event on the RTC WakeUp Timer associated Exti line. Return value: • None.
<code>__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_EVENT</code>	Description: • Disable event on the RTC WakeUp Timer associated Exti line. Return value: • None.
<code>__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_FALLING_EDGE</code>	Description: • Enable falling edge trigger on the RTC WakeUp Timer associated Exti line. Return value: • None.
<code>__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_FALLING_EDGE</code>	Description: • Disable falling edge trigger on the RTC WakeUp Timer associated Exti line. Return value: • None.
<code>__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_EDGE</code>	Description: • Enable rising edge trigger on the RTC WakeUp Timer associated Exti line. Return value: • None.
<code>__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_EDGE</code>	Description: • Disable rising edge trigger on the RTC WakeUp Timer

	associated Exti line.
	Return value:
	• None.
<code>__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_FALLING_EDGE</code>	Description:
	• Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.
	Return value:
	• None.
<code>__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_FALLING_EDGE</code>	Description:
	• Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.
	Return value:
	• None.
<code>__HAL_RTC_WAKEUPTIMER_EXTI_GET_FLAG</code>	Description:
	• Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.
	Return value:
	• Line: Status.
<code>__HAL_RTC_WAKEUPTIMER_EXTI_CLEAR_FLAG</code>	Description:
	• Clear the RTC WakeUp Timer associated Exti line flag.
	Return value:
	• None.
<code>__HAL_RTC_WAKEUPTIMER_EXTI_GENERATE_SWIT</code>	Description:
	• Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.
	Return value:
	• None.
RTCEx Wakeup Timer Definitions	
<code>RTC_WAKEUPCLOCK_RTCCLK_DIV16</code>	
<code>RTC_WAKEUPCLOCK_RTCCLK_DIV8</code>	

RTC_WAKEUPCLOCK_RTCCLK_DIV4
RTC_WAKEUPCLOCK_RTCCLK_DIV2
RTC_WAKEUPCLOCK_CK_SPRE_16BITS
RTC_WAKEUPCLOCK_CK_SPRE_17BITS

40 HAL SMARTCARD Generic Driver

40.1 SMARTCARD Firmware driver registers structures

40.1.1 SMARTCARD_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*
- *uint32_t OneBitSampling*
- *uint32_t Prescaler*
- *uint32_t GuardTime*
- *uint32_t NACKState*
- *uint32_t TimeOutEnable*
- *uint32_t TimeOutValue*
- *uint32_t BlockLength*
- *uint32_t AutoRetryCount*

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate***
Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hsc->Init.BaudRate)))
- ***uint32_t SMARTCARD_InitTypeDef::WordLength***
Specifies the number of data bits transmitted or received in a frame. This parameter [SMARTCARD_Word_Length](#) can only be set to 9 (8 data + 1 parity bits).
- ***uint32_t SMARTCARD_InitTypeDef::StopBits***
Specifies the number of stop bits [SMARTCARD_Stop_Bits](#). Only 0.5 or 1.5 stop bits are authorized in SmartCard mode.
- ***uint32_t SMARTCARD_InitTypeDef::Parity***
Specifies the parity mode. This parameter can be a value of [SMARTCARD_Parity](#)
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.
- ***uint32_t SMARTCARD_InitTypeDef::Mode***
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [SMARTCARD_Mode](#)
- ***uint32_t SMARTCARD_InitTypeDef::CLKPolarity***
Specifies the steady state of the serial clock. This parameter can be a value of [SMARTCARD_Clock_Polarity](#)
- ***uint32_t SMARTCARD_InitTypeDef::CLKPhase***
Specifies the clock transition on which the bit capture is made. This parameter can be a value of [SMARTCARD_Clock_Phase](#)

- ***uint32_t SMARTCARD_InitTypeDef::CLKLastBit***
Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of [SMARTCARD_Last_Bit](#)
- ***uint32_t SMARTCARD_InitTypeDef::OneBitSampling***
Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of [SMARTCARD_OneBit_Sampling](#)
- ***uint32_t SMARTCARD_InitTypeDef::Prescaler***
Specifies the SmartCard Prescaler
- ***uint32_t SMARTCARD_InitTypeDef::GuardTime***
Specifies the SmartCard Guard Time
- ***uint32_t SMARTCARD_InitTypeDef::NACKState***
Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of [SMARTCARD_NACK_Enable](#)
- ***uint32_t SMARTCARD_InitTypeDef::TimeoutEnable***
Specifies whether the receiver timeout is enabled. This parameter can be a value of [SMARTCARD_Timeout_Enable](#)
- ***uint32_t SMARTCARD_InitTypeDef::TimeoutValue***
Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- ***uint32_t SMARTCARD_InitTypeDef::BlockLength***
Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- ***uint32_t SMARTCARD_InitTypeDef::AutoRetryCount***
Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)

40.1.2 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](#)

40.1.3 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***
- ***SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init***
- ***SMARTCARD_AdvFeatureInitTypeDef SMARTCARD_HandleTypeDef::AdvancedInit***
- ***uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr***
- ***uint16_t SMARTCARD_HandleTypeDef::TxXferSize***
- ***uint16_t SMARTCARD_HandleTypeDef::TxXferCount***
- ***uint8_t* SMARTCARD_HandleTypeDef::pRxBuffPtr***
- ***uint16_t SMARTCARD_HandleTypeDef::RxXferSize***
- ***uint16_t SMARTCARD_HandleTypeDef::RxXferCount***
- ***DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmatx***

- `DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmarx`
- `HAL_LockTypeDef SMARTCARD_HandleTypeDef::Lock`
- `__IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::State`
- `__IO uint32_t SMARTCARD_HandleTypeDef::ErrorCode`

40.2 SMARTCARD Firmware driver API description

40.2.1 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

- These parameters can be configured:
 - Baud Rate
 - Parity: parity should be enabled, Frame Length is fixed to 8 bits plus parity: the USART frame format is given in the table 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 18: USART frame formats

M1 M0 bits	PCE bit	USART frame
01	1	SB 8 bit data PB STB

The `HAL_SMARTCARD_Init()` API follow respectively the USART (a)synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [`HAL\_SMARTCARD\_Init\(\)`](#)
- [`HAL\_SMARTCARD\_DeInit\(\)`](#)
- [`HAL\_SMARTCARD\_MspInit\(\)`](#)
- [`HAL\_SMARTCARD\_MspDeInit\(\)`](#)

40.2.2 IO operation functions

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\(\)](#)

40.2.3 Peripheral State functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL_SMARTCARD_GetState() API is helpful to check in run-time the state of the SMARTCARD peripheral
- SMARTCARD_SetConfig() API configures the SMARTCARD peripheral
- SMARTCARD_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization

This section contains the following APIs:

- [HAL_SMARTCARD_GetState\(\)](#)
- [HAL_SMARTCARD_GetError\(\)](#)

40.2.4 HAL_SMARTCARD_Init

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsc)
Function Description	Initializes the SMARTCARD mode according to the specified parameters in the SMARTCARD_InitTypeDef and creates the associated handle .
Parameters	• hsc: SMARTCARD handle
Return values	• HAL: status

40.2.5 HAL_SMARTCARD_DeInit

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsc)
Function Description	DeInitializes the SMARTCARD peripheral.
Parameters	• hsc: SMARTCARD handle
Return values	• HAL: status

40.2.6 HAL_SMARTCARD_MspInit

Function Name	void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsc)
Function Description	SMARTCARD MSP Init.
Parameters	• hsc: SMARTCARD handle
Return values	• None:

40.2.7 HAL_SMARTCARD_MspDeInit

Function Name	void HAL_SMARTCARD_MspDeInit

(SMARTCARD_HandleTypeDef * hsc)

Function Description SMARTCARD MSP DeInit.

Parameters

- **hsc:** SMARTCARD handle

Return values

- **None:**

40.2.8 HAL_SMARTCARD_Transmit

Function Name **HAL_StatusTypeDef HAL_SMARTCARD_Transmit**
(SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function Description Send an amount of data in blocking mode.

Parameters

- **hsc:** SMARTCARD handle
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

40.2.9 HAL_SMARTCARD_Receive

Function Name **HAL_StatusTypeDef HAL_SMARTCARD_Receive**
(SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function Description Receive an amount of data in blocking mode.

Parameters

- **hsc:** SMARTCARD handle
- **pData:** pointer to data buffer
- **Size:** amount of data to be received
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

40.2.10 HAL_SMARTCARD_Transmit_IT

Function Name **HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT**
(SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)

Function Description Send an amount of data in interrupt mode.

Parameters

- **hsc:** SMARTCARD handle
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

40.2.11 HAL_SMARTCARD_Receive_IT

Function Name **HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT**
(SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)

Function Description Receive an amount of data in interrupt mode.

Parameters	• hsc: SMARTCARD handle • pData: pointer to data buffer • Size: amount of data to be received
Return values	• HAL: status

40.2.12 HAL_SMARTCARD_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in DMA mode.
Parameters	• hsc: SMARTCARD handle • pData: pointer to data buffer • Size: amount of data to be sent
Return values	• HAL: status

40.2.13 HAL_SMARTCARD_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in DMA mode.
Parameters	• hsc: SMARTCARD handle • pData: pointer to data buffer • Size: amount of data to be received
Return values	• HAL: status
Notes	• The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)

40.2.14 HAL_SMARTCARD_IRQHandler

Function Name	void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsc)
Function Description	SMARTCARD interrupt requests handling.
Parameters	• hsc: SMARTCARD handle
Return values	• None:

40.2.15 HAL_SMARTCARD_TxCpltCallback

Function Name	void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsc)
Function Description	Tx Transfer completed callbacks.
Parameters	• hsc: SMARTCARD handle
Return values	• None:

40.2.16 HAL_SMARTCARD_RxCpltCallback

Function Name	void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsc)
Function Description	Rx Transfer completed callbacks.
Parameters	• hsc: SMARTCARD handle
Return values	• None:

40.2.17 HAL_SMARTCARD_ErrorCallback

Function Name	void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsc)
Function Description	SMARTCARD error callbacks.
Parameters	• hsc: SMARTCARD handle
Return values	• None:

40.2.18 HAL_SMARTCARD_GetState

Function Name	HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDef * hsc)
Function Description	return the SMARTCARD state
Parameters	• hsc: SMARTCARD handle
Return values	• HAL: state

40.2.19 HAL_SMARTCARD_GetError

Function Name	uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsc)
Function Description	Return the SMARTCARD error code.
Parameters	• hsc: : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD.
Return values	• SMARTCARD: Error Code

40.3 SMARTCARD Firmware driver defines

40.3.1 SMARTCARD

SMARTCARD Advanced Features Initialization

SMARTCARD_ADVFEATURE_NO_INIT
 SMARTCARD_ADVFEATURE_TXINVERT_INIT
 SMARTCARD_ADVFEATURE_RXINVERT_INIT
 SMARTCARD_ADVFEATURE_DATAINVERT_INIT
 SMARTCARD_ADVFEATURE_SWAP_INIT
 SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT

SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT

SMARTCARD_ADVFEATURE_MSBFIRST_INIT

IS_SMARTCARD_ADVFEATURE_INIT

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD_PHASE_2EDGE

IS_SMARTCARD_PHASE

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD_POLARITY_HIGH

IS_SMARTCARD_POLARITY

SMARTCARD CR3 LSB Position

SMARTCARD_CR3_SCARCNT_LSB_POS

SMARTCARD Data Inv

SMARTCARD_ADVFEATURE_DATAINV_DISABLE

SMARTCARD_ADVFEATURE_DATAINV_ENABLE

IS_SMARTCARD_ADVFEATURE_DATAINV

SMARTCARD DMA on Rx Error

SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR

SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR

IS_SMARTCARD_ADVFEATURE_DMAONRXERROR

SMARTCARD DMA Requests

SMARTCARD_DMAREQ_TX

SMARTCARD_DMAREQ_RX

SMARTCARD Error Code

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

__HAL_SMARTCARD_RESET_HANDLE
_STATE

Description:

- Reset SMARTCARD handle state.

Parameters:

- __HANDLE__: specifies the

SMARTCARD Handle. The Handle Instance which can be USART1 or USART2

Return value:

- None

`__HAL_SMARTCARD_FLUSH_DRREGISTER`

Description:

- Flushes the Smartcard DR register.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

`__HAL_SMARTCARD_CLEAR_FLAG`

Description:

- Clears 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
 - SMARTCARD_CLEAR_FEF
 - SMARTCARD_CLEAR_NEF
 - SMARTCARD_CLEAR_OREF
 - SMARTCARD_CLEAR_IDLEF
 - SMARTCARD_CLEAR_TCF
 - SMARTCARD_CLEAR_RTOF
 - SMARTCARD_CLEAR_EOBF

Return value:

- None

`__HAL_SMARTCARD_CLEAR_PEF`

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

`__HAL_SMARTCARD_CLEAR_FEF`

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`**Description:**

- Clear the SMARTCARD ORE pending flag.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

`__HAL_SMARTCARD_CLEAR_IDLEFLAG`**Description:**

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

`__HAL_SMARTCARD_GET_FLAG`**Description:**

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

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - SMARTCARD_FLAG_REACK: Receive enable acknowledge flag
 - SMARTCARD_FLAG_TEACK: Transmit enable acknowledge flag
 - SMARTCARD_FLAG_BUSY: Busy

- flag
- SMARTCARD_FLAG_EOBF: End of block flag
- SMARTCARD_FLAG_RTOF: Receiver timeout flag
- SMARTCARD_FLAG_TXE: Transmit data register empty flag
- SMARTCARD_FLAG_TC: Transmission Complete flag
- SMARTCARD_FLAG_RXNE: Receive data register not empty flag
- SMARTCARD_FLAG_IDLE: Idle line detection flag
- SMARTCARD_FLAG_ORE: OverRun Error flag
- SMARTCARD_FLAG_NE: Noise Error flag
- SMARTCARD_FLAG_FE: Framing Error flag
- SMARTCARD_FLAG_PE: Parity Error flag

Return value:

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

Description:

- Enables the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- __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

__HAL_SMARTCARD_ENABLE_IT

interrupt(Frame error, noise error, overrun error)

`__HAL_SMARTCARD_DISABLE_IT`

Return value:

- None

Description:

- Disables the specified SmartCard interrupt.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- `__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_GET_IT`

Description:

- Checks whether the specified SmartCard interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- `__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

Description:

- Checks whether the specified SmartCard interrupt interrupt source is enabled.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- __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

`__HAL_SMARTCARD_CLEAR_IT`**Return value:**

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

Description:

- Clears the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- `__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 which can be USART1 or USART2.
- `__REQ__`: specifies the request flag to set This parameter can be one of the following values:
 - `SMARTCARD_RXDATA_FLUSH_REQUEST`: Receive Data flush

- Request
- SMARTCARD_TXDATA_FLUSH_REQUEST: Transmit data flush Request

Return value:

- None

`__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE`

Description:

- Enables the SMARTCARD one bit sample method.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

`__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE`

Description:

- Disables the SMARTCARD one bit sample method.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

`__HAL_SMARTCARD_ENABLE`

Description:

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.

Return value:

- None

`__HAL_SMARTCARD_DISABLE`

Description:

- Disable the USART associated to the SMARTCARD Handle.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.

Return value:

`__HAL_SMARTCARD_DMA_REQUEST_ENABLE`

- None

Description:

- Macros to enable or disable the SmartCard DMA request.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- `__REQUEST__`: specifies the SmartCard DMA request. This parameter can be one of the following values:
 - `SMARTCARD_DMAREQ_TX`: SmartCard DMA transmit request
 - `SMARTCARD_DMAREQ_RX`: SmartCard DMA receive request

`__HAL_SMARTCARD_DMA_REQUEST_DISABLE`

`IS_SMARTCARD_BAUDRATE`

Description:

- Check the Baud rate range.

Parameters:

- `__BAUDRATE__`: Baud rate set by the configuration function.

Return value:

- Test: result (TRUE or FALSE)

`IS_SMARTCARD_BLOCKLENGTH`

Description:

- Check the block length range.

Parameters:

- `__LENGTH__`: block length.

Return value:

- Test: result (TRUE or FALSE)

`IS_SMARTCARD_TIMEOUT_VALUE`

Description:

- Check the receiver timeout value.

Parameters:

- `__TIMEOUTVALUE__`: receiver timeout value.

Return value:

- Test: result (TRUE or FALSE)

`IS_SMARTCARD_AUTORETRY_COUNTER`

Description:

- Check the SMARTCARD autoretry counter value.

Parameters:

- `__COUNT__`: number of retransmissions

Return value:

- Test: result (TRUE or FALSE)

SMARTCARD Flags

<code>SMARTCARD_FLAG_REACK</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 GTPR GT LSB Position

`SMARTCARD_GTPR_GT_LSB_POS`

SMARTCARD Interruption Mask

`SMARTCARD_IT_MASK`

SMARTCARD Interrupt 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 IT CLEAR Flags

`SMARTCARD_CLEAR_PEF` Parity Error Clear Flag

SMARTCARD_CLEAR_FEF	Framing Error Clear Flag
SMARTCARD_CLEAR_NEF	Noise detected Clear Flag
SMARTCARD_CLEAR_OREF	OverRun Error Clear Flag
SMARTCARD_CLEAR_IDLEF	IDLE line detected Clear Flag
SMARTCARD_CLEAR_TCF	Transmission Complete Clear Flag
SMARTCARD_CLEAR_RTOF	Receiver Time Out Clear Flag
SMARTCARD_CLEAR_EOBF	End Of Block Clear Flag

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE
SMARTCARD_LASTBIT_ENABLE
IS_SMARTCARD_LASTBIT

SMARTCARD Mode

SMARTCARD_MODE_RX
SMARTCARD_MODE_TX
SMARTCARD_MODE_TX_RX
IS_SMARTCARD_MODE

SMARTCARD MSB First

SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE
SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE
IS_SMARTCARD_ADVFEATURE_MSBFIRST

SMARTCARD NACK Enable

SMARTCARD_NACK_ENABLE
SMARTCARD_NACK_DISABLE
IS_SMARTCARD_NACK

SMARTCARD OneBit Sampling

SMARTCARD_ONE_BIT_SAMPLE_DISABLE
SMARTCARD_ONE_BIT_SAMPLE_ENABLE
IS_SMARTCARD_ONE_BIT_SAMPLE

SMARTCARD Overrun Enabling

SMARTCARD_ADVFEATURE_OVERRUN_ENABLE
SMARTCARD_ADVFEATURE_OVERRUN_DISABLE
IS_SMARTCARD_OVERRUN

SMARTCARD Parity

SMARTCARD_PARITY_EVEN
SMARTCARD_PARITY_ODD
IS_SMARTCARD_PARITY

SMARTCARD Request Parameters

SMARTCARD_RXDATA_FLUSH_REQUEST Receive Data flush Request

SMARTCARD_TXDATA_FLUSH_REQUEST Transmit data flush Request

IS_SMARTCARD_REQUEST_PARAMETER

SMARTCARD RTOR BLEN LSB Position

SMARTCARD_RTOR_BLEN_LSB_POS

SMARTCARD Rx Inv

SMARTCARD_ADVFEATURE_RXINV_DISABLE

SMARTCARD_ADVFEATURE_RXINV_ENABLE

IS_SMARTCARD_ADVFEATURE_RXINV

SMARTCARD Rx Tx Swap

SMARTCARD_ADVFEATURE_SWAP_DISABLE

SMARTCARD_ADVFEATURE_SWAP_ENABLE

IS_SMARTCARD_ADVFEATURE_SWAP

SMARTCARD Stop Bits

SMARTCARD_STOPBITS_0_5

SMARTCARD_STOPBITS_1_5

IS_SMARTCARD_STOPBITS

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DISABLE

SMARTCARD_TIMEOUT_ENABLE

IS_SMARTCARD_TIMEOUT

SMARTCARD Tx Inv

SMARTCARD_ADVFEATURE_TXINV_DISABLE

SMARTCARD_ADVFEATURE_TXINV_ENABLE

IS_SMARTCARD_ADVFEATURE_TXINV

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B

IS_SMARTCARD_WORD_LENGTH

41 HAL SMARTCARD Extension Driver

41.1 SMARTCARDEx Firmware driver API description

41.1.1 How to use this driver

The Extended SMARTCARD HAL driver can be used as follow:

1. After having configured the SMARTCARD basic features with `HAL_SMARTCARD_Init()`, then if required, program SMARTCARD advanced features (TX/RX pins swap, `TimeOut`, auto-retry counter,...) in the `hsc AdvancedInit` structure.

41.1.2 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\(\)`](#)

41.1.3 HAL_SMARTCARDEx_BlockLength_Config

Function Name	<code>void HAL_SMARTCARDEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsc, uint8_t BlockLength)</code>
Function Description	Update on the fly the SMARTCARD block length in RTOR register.
Parameters	• hsc: SMARTCARD handle • BlockLength: SMARTCARD block length (8-bit long at most)
Return values	• None:

41.1.4 HAL_SMARTCARDEx_TimeOut_Config

Function Name	<code>void HAL_SMARTCARDEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsc, uint32_t TimeOutValue)</code>
Function Description	Update on the fly the receiver timeout value in RTOR register.
Parameters	• hsc: SMARTCARD handle • TimeOutValue: receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0xFFFFFFFF.

Return values • **None:**

41.1.5 HAL_SMARTCARDEx_EnableReceiverTimeout

Function Name HAL_StatusTypeDef
 HAL_SMARTCARDEx_EnableReceiverTimeout
 (SMARTCARD_HandleTypeDef * hsc)

Function Description Enable the SMARTCARD receiver timeout feature.

Parameters • **hsc:** SMARTCARD handle

Return values • **HAL:** status

41.1.6 HAL_SMARTCARDEx_DisableReceiverTimeout

Function Name HAL_StatusTypeDef
 HAL_SMARTCARDEx_DisableReceiverTimeout
 (SMARTCARD_HandleTypeDef * hsc)

Function Description Disable the SMARTCARD receiver timeout feature.

Parameters • **hsc:** SMARTCARD handle

Return values • **HAL:** status

41.2 SMARTCARDEx Firmware driver defines

41.2.1 SMARTCARDEx

SMARTCARDEx Exported Macros

SMARTCARD_GETCLOCKSOURCE **Description:**

- Reports the SMARTCARD clock source.

Parameters:

- **__HANDLE__:** specifies the USART Handle
- **__CLOCKSOURCE__:** : output variable

Return value:

- the: USART clocking source, written in **__CLOCKSOURCE__**.

42 HAL SMBUS Generic Driver

42.1 SMBUS Firmware driver registers structures

42.1.1 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 Peripheral is selected. This parameter can be a value of [SMBUS_peripheral_mode](#)
- ***uint32_t SMBUS_InitTypeDef::SMBusTimeout***
Specifies the content of the 32 Bits SMBUS_TIMEOUT_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

42.1.2 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 tate
- ***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 , see [SMBUS_Error_Code](#)

42.2 SMBUS Firmware driver API description

42.2.1 Initialization and de-initialization functions

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

- User must Implement HAL_SMBUS_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC).
- Call the function HAL_SMBUS_Init() to configure the selected device with the selected configuration:
 - Clock Timing
 - Bus Timeout
 - Analog Filter mode
 - Own Address 1
 - Addressing mode (Master, Slave)
 - Dual Addressing mode
 - Own Address 2
 - Own Address 2 Mask
 - General call mode
 - Nostretch mode
 - Packet Error Check mode
 - Peripheral mode
- Call the function HAL_SMBUS_DeInit() to restore the default configuration of the selected SMBUSx peripheral.

This section contains the following APIs:

- [*HAL_SMBUS_Init\(\)*](#)
- [*HAL_SMBUS_DeInit\(\)*](#)
- [*HAL_SMBUS_MspInit\(\)*](#)
- [*HAL_SMBUS_MspDeInit\(\)*](#)

42.2.2 IO operation functions

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

1. Blocking mode function to check if device is ready for usage is :
 - HAL_SMBUS_IsDeviceReady()
2. There is only one mode of transfer:
 - No-Blocking mode : The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. No-Blocking mode functions with Interrupt are :
 - HAL_SMBUS_Master_Transmit_IT()
 - HAL_SMBUS_Master_Receive_IT()
 - HAL_SMBUS_Slave_Transmit_IT()
 - HAL_SMBUS_Slave_Receive_IT()
 - HAL_SMBUS_EnableListen_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\(\)](#)
- [HAL_SMBUS_EV_IRQHandler\(\)](#)
- [HAL_SMBUS_ER_IRQHandler\(\)](#)
- [HAL_SMBUS_MasterTxCpltCallback\(\)](#)
- [HAL_SMBUS_MasterRxCpltCallback\(\)](#)
- [HAL_SMBUS_SlaveTxCpltCallback\(\)](#)
- [HAL_SMBUS_SlaveRxCpltCallback\(\)](#)
- [HAL_SMBUS_AddrCallback\(\)](#)
- [HAL_SMBUS_ListenCpltCallback\(\)](#)
- [HAL_SMBUS_ErrorCallback\(\)](#)

42.2.3 Peripheral State and Errors 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_SMBUS_GetState\(\)](#)
- [HAL_SMBUS_GetError\(\)](#)

42.2.4 HAL_SMBUS_Init

Function Name	HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)
Function Description	Initializes the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and create the associated handle.
Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• HAL: status

42.2.5 HAL_SMBUS_DeInit

Function Name	HAL_StatusTypeDef HAL_SMBUS_DeInit (SMBUS_HandleTypeDef * hsmbus)
Function Description	DeInitializes the SMBUS peripheral.

Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• HAL: status

42.2.6 HAL_SMBUS_Msplnit

Function Name	void HAL_SMBUS_Msplnit (SMBUS_HandleTypeDef * hsmbus)
Function Description	SMBUS MSP Init.
Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None:

42.2.7 HAL_SMBUS_MspDeInit

Function Name	void HAL_SMBUS_MspDeInit (SMBUS_HandleTypeDef * hsmbus)
Function Description	SMBUS MSP DeInit.
Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None:

42.2.8 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	• 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 Transfer Request Definition
Return values	• HAL: status

42.2.9 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.

blocking mode with Interrupt.

Parameters	• 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 Transfer Request Definition
Return values	• HAL: status

42.2.10 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	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • DevAddress: Target device address
Return values	• HAL: status
Notes	• : This abort can be called only if state is ready

42.2.11 HAL_SMBUS_Slave_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)
Function Description	Transmit in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.
Parameters	• 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 Transfer Request Definition
Return values	• HAL: status

42.2.12 HAL_SMBUS_Slave_Receive_IT

Function Name	HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)
Function Description	Receive in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.
Parameters	• 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 Transfer Request Definition

Return values

- **HAL:** status

42.2.13 HAL_SMBUS_EnableListen_IT

Function Name

HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function Description

This function enable the Address listen mode in Slave mode.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

42.2.14 HAL_SMBUS_DisableListen_IT

Function Name

HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function Description

This function disable the Address listen mode.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

42.2.15 HAL_SMBUS_EnableAlert_IT

Function Name

HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function Description

Enable SMBUS alert.

Parameters

- **hsmbus:** : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

42.2.16 HAL_SMBUS_DisableAlert_IT

Function Name

HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function Description

Disable SMBUS alert.

Parameters

- **hsmbus:** : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

42.2.17 HAL_SMBUS_IsDeviceReady

Function Name	HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)
Function Description	Checks if target device is ready for communication.
Parameters	• 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	• HAL: status
Notes	• This function is used with Memory devices

42.2.18 HAL_SMBUS_EV_IRQHandler

Function Name	void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)
Function Description	This function handles SMBUS event interrupt request.
Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None:

42.2.19 HAL_SMBUS_ER_IRQHandler

Function Name	void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)
Function Description	This function handles SMBUS error interrupt request.
Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None:

42.2.20 HAL_SMBUS_MasterTxCpltCallback

Function Name	void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Master Tx Transfer completed callbacks.
Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None:

42.2.21 HAL_SMBUS_MasterRxCpltCallback

Function Name	void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Master Rx Transfer completed callbacks.
Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None:

42.2.22 HAL_SMBUS_SlaveTxCpltCallback

Function Name	void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Slave Tx Transfer completed callbacks.
Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None:

42.2.23 HAL_SMBUS_SlaveRxCpltCallback

Function Name	void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Slave Rx Transfer completed callbacks.
Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None:

42.2.24 HAL_SMBUS_AddrCallback

Function Name	void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)
Function Description	Slave Address Match callbacks.
Parameters	• 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	• None:

42.2.25 HAL_SMBUS_ListenCpltCallback

Function Name	void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)
Function Description	Slave Listen Complete callbacks.
Parameters	• hsmbus: : Pointer to a SMBUS_HandleTypeDef structure

that contains the configuration information for the specified SMBUS.

Return values

- **None:**

42.2.26 HAL_SMBUS_ErrorCallback

Function Name **void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef *hsmbus)**

Function Description SMBUS error callbacks.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

42.2.27 HAL_SMBUS_GetState

Function Name **uint32_t HAL_SMBUS_GetState (SMBUS_HandleTypeDef *hsmbus)**

Function Description Returns the SMBUS state.

Parameters

- **hsmbus:** : SMBUS handle

Return values

- **HAL:** state

42.2.28 HAL_SMBUS_GetError

Function Name **uint32_t HAL_SMBUS_GetError (SMBUS_HandleTypeDef *hsmbus)**

Function Description Return the SMBUS error code.

Parameters

- **hsmbus:** : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **SMBUS:** Error Code

42.3 SMBUS Firmware driver defines

42.3.1 SMBUS

SMBUS Addressing Mode

SMBUS_ADDRESSINGMODE_7BIT

SMBUS_ADDRESSINGMODE_10BIT

IS_SMBUS_ADDRESSING_MODE

SMBUS Analog Filter

SMBUS_ANALOGFILTER_ENABLE

SMBUS_ANALOGFILTER_DISABLE

IS_SMBUS_ANALOG_FILTER

SMBUS Dual Addressing Mode

SMBUS_DUALADDRESS_DISABLE

SMBUS_DUALADDRESS_ENABLE

IS_SMBUS_DUAL_ADDRESS

SMBUS Error Code

HAL_SMBUS_ERROR_NONE	No error
HAL_SMBUS_ERROR_BERR	BERR error
HAL_SMBUS_ERROR_ARLO	ARLO error
HAL_SMBUS_ERROR_ACKF	ACKF error
HAL_SMBUS_ERROR_OVR	OVR error
HAL_SMBUS_ERROR_HALTIMEOUT	Timeout error
HAL_SMBUS_ERROR_BUSTIMEOUT	Bus Timeout error
HAL_SMBUS_ERROR_ALERT	Alert error
HAL_SMBUS_ERROR_PECERR	PEC error

IO operation functions

HAL_SMBUS_EnableListen_IT

HAL_SMBUS_AddrCallback

HAL_SMBUS_ListenCpltCallback

SMBUS Exported Macros

__HAL_SMBUS_RESET_HANDLE_STATE

Description:

- Reset SMBUS handle state.

Parameters:

- __HANDLE__**: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.

Return value:

- None

__HAL_SMBUS_ENABLE_IT

Description:

- Enable or disable the specified SMBUS interrupts.

Parameters:

- __HANDLE__**: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.
- __INTERRUPT__**: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - SMBUS_IT_ERRI: Errors interrupt enable

- SMBUS_IT_TCI: Transfer complete interrupt enable
- SMBUS_IT_STOPI: STOP detection interrupt enable
- SMBUS_IT_NACKI: NACK received interrupt enable
- SMBUS_IT_ADDRI: Address match interrupt enable
- SMBUS_IT_RXI: RX interrupt enable
- SMBUS_IT_TXI: TX interrupt enable

Return value:

- None

__HAL_SMBUS_DISABLE_IT

__HAL_SMBUS_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.
- __INTERRUPT__: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
 - SMBUS_IT_ERRI: Errors interrupt enable
 - SMBUS_IT_TCI: Transfer complete interrupt enable
 - SMBUS_IT_STOPI: STOP detection interrupt enable
 - SMBUS_IT_NACKI: NACK received interrupt enable
 - SMBUS_IT_ADDRI: Address match interrupt enable
 - SMBUS_IT_RXI: RX interrupt enable
 - SMBUS_IT_TXI: TX interrupt enable

Return value:

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

SMBUS_FLAG_MASK

Description:

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

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `SMBUS_FLAG_TXE`: Transmit data register empty
 - `SMBUS_FLAG_TXIS`: Transmit interrupt status
 - `SMBUS_FLAG_RXNE`: Receive data register not empty
 - `SMBUS_FLAG_ADDR`: Address matched (slave mode)
 - `SMBUS_FLAG_AF`: NACK received flag
 - `SMBUS_FLAG_STOPF`: STOP detection flag
 - `SMBUS_FLAG_TC`: Transfer complete (master mode)
 - `SMBUS_FLAG_TCR`: Transfer complete reload
 - `SMBUS_FLAG_BERR`: Bus error
 - `SMBUS_FLAG_ARLO`: Arbitration lost
 - `SMBUS_FLAG_OVR`: Overrun/Underrun
 - `SMBUS_FLAG_PECERR`: PEC error in reception
 - `SMBUS_FLAG_TIMEOUT`: Timeout or Tlow detection flag
 - `SMBUS_FLAG_ALERT`: SMBus alert
 - `SMBUS_FLAG_BUSY`: Bus busy
 - `SMBUS_FLAG_DIR`: Transfer direction (slave mode)

Return value:

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

`__HAL_SMBUS_GET_FLAG``__HAL_SMBUS_CLEAR_FLAG`**Description:**

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

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle. This parameter can be SMBUSx where x: 1 or 2 to select the SMBUS peripheral.
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination

of the following values:

- SMBUS_FLAG_ADDR: Address matched (slave mode)
- SMBUS_FLAG_AF: NACK received flag
- SMBUS_FLAG_STOPF: STOP detection flag
- SMBUS_FLAG_BERR: Bus error
- SMBUS_FLAG_ARLO: Arbitration lost
- SMBUS_FLAG_OVR: Overrun/Underrun
- SMBUS_FLAG_PECERR: PEC error in reception
- SMBUS_FLAG_TIMEOUT: Timeout or Tlow detection flag
- SMBUS_FLAG_ALERT: SMBus alert

Return value:

- None

__HAL_SMBUS_ENABLE
 __HAL_SMBUS_DISABLE
 __SMBUS_RESET_CR1
 __SMBUS_RESET_CR2
 __SMBUS_GENERATE_START
 __SMBUS_GET_ADDR_MATCH
 __SMBUS_GET_DIR
 __SMBUS_GET_STOP_MODE
 __SMBUS_GET_PEC_MODE
 __SMBUS_GET_ALERT_ENABLE
 __HAL_SMBUS_GENERATE_NACK
 IS_SMBUS_OWN_ADDRESS1
 IS_SMBUS_OWN_ADDRESS2

SMBUS Flag Definition

SMBUS_FLAG_TXE
 SMBUS_FLAG_TXIS
 SMBUS_FLAG_RXNE
 SMBUS_FLAG_ADDR
 SMBUS_FLAG_AF
 SMBUS_FLAG_STOPF
 SMBUS_FLAG_TC
 SMBUS_FLAG_TCR

SMBUS_FLAG_BERR
SMBUS_FLAG_ARLO
SMBUS_FLAG_OVR
SMBUS_FLAG_PECERR
SMBUS_FLAG_TIMEOUT
SMBUS_FLAG_ALERT
SMBUS_FLAG_BUSY
SMBUS_FLAG_DIR

SMBUS General Call Enabling

SMBUS_GENERALCALL_DISABLE
SMBUS_GENERALCALL_ENABLE
IS_SMBUS_GENERAL_CALL

SMBUS Interrupt Configuration Definition

SMBUS_IT_ERRI
SMBUS_IT_TCI
SMBUS_IT_STOPI
SMBUS_IT_NACKI
SMBUS_IT_ADDRI
SMBUS_IT_RXI
SMBUS_IT_TXI
SMBUS_IT_TX
SMBUS_IT_RX
SMBUS_IT_ALERT
SMBUS_IT_ADDR

SMBUS Nostretch Enabling

SMBUS_NOSTRETCH_DISABLE
SMBUS_NOSTRETCH_ENABLE
IS_SMBUS_NO_STRETCH

SMBUS Own Address2 Masks

SMBUS_OA2_NOMASK
SMBUS_OA2_MASK01
SMBUS_OA2_MASK02
SMBUS_OA2_MASK03
SMBUS_OA2_MASK04
SMBUS_OA2_MASK05
SMBUS_OA2_MASK06

SMBUS_OA2_MASK07

IS_SMBUS_OWN_ADDRESS2_MASK

SMBUS Packet Error Check Enabling

SMBUS_PEC_DISABLE

SMBUS_PEC_ENABLE

IS_SMBUS_PEC

SMBUS Peripheral Mode

SMBUS_PERIPHERAL_MODE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP

IS_SMBUS_PERIPHERAL_MODE

SMBUS Mode Definition

SMBUS_SOFTEND_MODE

SMBUS_RELOAD_MODE

SMBUS_AUTOEND_MODE

SMBUS_SENDPEC_MODE

IS_SMBUS_TRANSFER_MODE

SMBUS StartStop Mode Definition

SMBUS_NO_STARTSTOP

SMBUS_GENERATE_STOP

SMBUS_GENERATE_START_READ

SMBUS_GENERATE_START_WRITE

IS_SMBUS_TRANSFER_REQUEST

SMBUS State

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

SMBUS Transfer Request Definition

SMBUS_FIRST_FRAME
SMBUS_NEXT_FRAME
SMBUS_FIRST_AND_LAST_FRAME_NO_PEC
SMBUS_LAST_FRAME_NO_PEC
SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC
SMBUS_LAST_FRAME_WITH_PEC
IS_SMBUS_TRANSFER_OPTIONS_REQUEST

43 HAL SPI Generic Driver

43.1 SPI Firmware driver registers structures

43.1.1 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 TMode*
- *uint32_t CRCCalculation*
- *uint32_t CRCPolynomial*

Field Documentation

- *uint32_t SPI_InitTypeDef::Mode*
Specifies the SPI operating mode. This parameter can be a value of [SPI_mode](#)
- *uint32_t SPI_InitTypeDef::Direction*
Specifies the SPI Directional mode state. This parameter can be a value of [SPI_Direction_mode](#)
- *uint32_t SPI_InitTypeDef::DataSize*
Specifies the SPI data size. This parameter can be a value of [SPI_data_size](#)
- *uint32_t SPI_InitTypeDef::CLKPolarity*
Specifies the serial clock steady state. This parameter can be a value of [SPI_Clock_Polarity](#)
- *uint32_t SPI_InitTypeDef::CLKPhase*
Specifies the clock active edge for the bit capture. This parameter can be a value of [SPI_Clock_Phase](#)
- *uint32_t SPI_InitTypeDef::NSS*
Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of [SPI_Slave_Select_management](#)
- *uint32_t SPI_InitTypeDef::BaudRatePrescaler*
Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of [SPI_BaudRate_Prescaler](#)
Note:The communication clock is derived from the master clock. The slave clock does not need to be set
- *uint32_t SPI_InitTypeDef::FirstBit*
Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of [SPI_MSB_LSB_transmission](#)
- *uint32_t SPI_InitTypeDef::TMode*
Specifies if the TI mode is enabled or not. This parameter can be a value of [SPI_TI_mode](#)

- ***uint32_t SPI_InitTypeDef::CRCCalculation***
Specifies if the CRC calculation is enabled or not. This parameter can be a value of [SPI_CRC_Calculation](#)
- ***uint32_t SPI_InitTypeDef::CRCPolynomial***
Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0 and Max_Data = 65535

43.1.2 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***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***void(* RxISR***
- ***void(* TxISR***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SPI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***SPI_TypeDef* SPI_HandleTypeDef::Instance***
SPI registers base address
- ***SPI_InitTypeDef SPI_HandleTypeDef::Init***
SPI communication parameters
- ***uint8_t* SPI_HandleTypeDef::pTxBuffPtr***
Pointer to SPI Tx transfer Buffer
- ***uint16_t SPI_HandleTypeDef::TxXferSize***
SPI Tx transfer size
- ***uint16_t SPI_HandleTypeDef::TxXferCount***
SPI Tx Transfer Counter
- ***uint8_t* SPI_HandleTypeDef::pRxBuffPtr***
Pointer to SPI Rx transfer Buffer
- ***uint16_t SPI_HandleTypeDef::RxXferSize***
SPI Rx transfer size
- ***uint16_t SPI_HandleTypeDef::RxXferCount***
SPI Rx Transfer Counter
- ***DMA_HandleTypeDef* SPI_HandleTypeDef::hdmatx***
SPI Tx DMA handle parameters
- ***DMA_HandleTypeDef* SPI_HandleTypeDef::hdmarx***
SPI Rx DMA handle parameters
- ***void(* SPI_HandleTypeDef::RxISR)(struct SPI_HandleTypeDef *hspi)***
function pointer on Rx ISR

- **`void(* SPI_HandleTypeDef::TxISR)(struct SPI_HandleTypeDef *hspi)`**
function pointer on Tx ISR
- **`HAL_LockTypeDef SPI_HandleTypeDef::Lock`**
SPI locking object
- **`__IO HAL_SPI_StateTypeDef SPI_HandleTypeDef::State`**
SPI communication state
- **`__IO uint32_t SPI_HandleTypeDef::ErrorCode`**
SPI Error code

43.2 SPI Firmware driver API description

43.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit ()API:
 - a. Enable the SPIx interface clock
 - b. SPI pins configuration
 - Enable the clock for the SPI GPIOs
 - Configure these SPI pins as alternate function push-pull
 - c. NVIC configuration if you need to use interrupt process
 - Configure the SPIx interrupt priority
 - Enable the NVIC SPI IRQ handle
 - d. DMA Configuration if you need to use DMA process
 - Declare a DMA_HandleTypeDef handle structure for the transmit or receive Channel
 - Enable the DMAx clock
 - Configure the DMA handle parameters
 - Configure the DMA Tx or Rx Channel
 - Associate the initialized hdma_tx(or _rx) handle to the hspi DMA Tx (or Rx) handle
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Channel
3. Program the Mode, Direction , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL_SPI_Init() API:
 - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit() API.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
 - a. Master 2Lines RxOnly
 - b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled
3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL_SPI_DMAPause()/ HAL_SPI_DMAStop() only under the SPI callbacks

Using the HAL it is not possible to reach all supported SPI frequency with the different SPI Modes, the following table resume the max SPI frequency reached with data size 8bits/16bits, according to frequency used on APBx Peripheral Clock (fPCLK) used by the SPI instance:

Table 19: Maximum SPI frequency vs data size (8 bits)

Process	Transfer mode	2 lines, full duplex		2 line, Rx only		1 line	
		Master	Slave	Master	Slave	Master	Slave
Tx/Rx	Polling	$f_{CPU}/4$	$f_{CPU}/8$	NA	NA	NA	NA
	Interrupt	$f_{CPU}/64$	$f_{CPU}/64$	NA	NA	NA	NA
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	NA	NA
Rx	Polling	$f_{CPU}/2$	$f_{CPU}/8$	$f_{CPU}/8$	$f_{CPU}/8$	$f_{CPU}/8$	$f_{CPU}/8$
	Interrupt	$f_{CPU}/64$	$f_{CPU}/32$	$f_{CPU}/32$	$f_{CPU}/16$	$f_{CPU}/32$	$f_{CPU}/32$
	DMA	$f_{CPU}/2$	$f_{CPU}/2$	$f_{CPU}/16$	$f_{CPU}/2$	$f_{CPU}/16$	$f_{CPU}/2$
Tx	Polling	$f_{CPU}/8$	$f_{CPU}/8$	NA	NA	$f_{CPU}/4$	$f_{CPU}/16$
	Interrupt	$f_{CPU}/8$	$f_{CPU}/32$	NA	NA	$f_{CPU}/8$	$f_{CPU}/16$
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	$f_{CPU}/2$	$f_{CPU}/2$

Table 20: Maximum SPI frequency vs data size (16 bits)

Process	Transfer mode	2 lines, full duplex		2 line, Rx only		1 line	
		Master	Slave	Master	Slave	Master	Slave
Tx/Rx	Polling	$f_{CPU}/4$	$f_{CPU}/8$	NA	NA	NA	NA
	Interrupt	$f_{CPU}/32$	$f_{CPU}/16$	NA	NA	NA	NA
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	NA	NA
Rx	Polling	$f_{CPU}/2$	$f_{CPU}/4$	$f_{CPU}/8$	$f_{CPU}/4$	$f_{CPU}/2$	$f_{CPU}/8$
	Interrupt	$f_{CPU}/32$	$f_{CPU}/8$	$f_{CPU}/16$	$f_{CPU}/16$	$f_{CPU}/16$	$f_{CPU}/8$
	DMA	$f_{CPU}/2$	$f_{CPU}/2$	$f_{CPU}/8$	$f_{CPU}/2$	$f_{CPU}/8$	$f_{CPU}/2$
Tx	Polling	$f_{CPU}/4$	$f_{CPU}/4$	NA	NA	$f_{CPU}/4$	$f_{CPU}/8$
	Interrupt	$f_{CPU}/4$	$f_{CPU}/16$	NA	NA	$f_{CPU}/8$	$f_{CPU}/8$
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	$f_{CPU}/2$	$f_{CPU}/2$

43.2.2 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
 - TMode

- CRC Calculation
- CRC Polynomial if CRC enabled
- 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_DeInit\(\)](#)
- [HAL_SPI_MspltInit\(\)](#)
- [HAL_SPI_MspltDeInit\(\)](#)

43.2.3 IO operation functions

The SPI supports master and slave mode :

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_SPI_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
 - HAL_SPI_Transmit() in 1Line (simplex) and 2Lines (full duplex) mode
 - HAL_SPI_Receive() in 1Line (simplex) and 2Lines (full duplex) mode
 - HAL_SPI_TransmitReceive() in full duplex mode
3. Non Blocking mode API's with Interrupt are :
 - HAL_SPI_Transmit_IT() in 1Line (simplex) and 2Lines (full duplex) mode
 - HAL_SPI_Receive_IT() in 1Line (simplex) and 2Lines (full duplex) mode
 - HAL_SPI_TransmitReceive_IT() in full duplex mode
 - HAL_SPI_IRQHandler()
4. Non Blocking mode functions with DMA are :
 - HAL_SPI_Transmit_DMA() in 1Line (simplex) and 2Lines (full duplex) mode
 - HAL_SPI_Receive_DMA() in 1Line (simplex) and 2Lines (full duplex) mode
 - HAL_SPI_TransmitReceive_DMA() in full duplex mode
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - HAL_SPI_TxCpltCallback()
 - HAL_SPI_RxCpltCallback()
 - HAL_SPI_TxRxCpltCallback()
 - HAL_SPI_TxHalfCpltCallback()
 - HAL_SPI_RxHalfCpltCallback()
 - HAL_SPI_TxRxHalfCpltCallback()
 - HAL_SPI_ErrorCallback()

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_DMABase\(\)](#)
- [HAL_SPI_DMAResume\(\)](#)
- [HAL_SPI_DMABaseStop\(\)](#)
- [HAL_SPI_IRQHandler\(\)](#)
- [HAL_SPI_TxCpltCallback\(\)](#)
- [HAL_SPI_RxCpltCallback\(\)](#)
- [HAL_SPI_TxRxCpltCallback\(\)](#)
- [HAL_SPI_TxHalfCpltCallback\(\)](#)
- [HAL_SPI_RxHalfCpltCallback\(\)](#)
- [HAL_SPI_TxRxHalfCpltCallback\(\)](#)
- [HAL_SPI_ErrorCallback\(\)](#)

43.2.4 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\(\)](#)

43.2.5 HAL_SPI_Init

Function Name	HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)
Function Description	Initializes the SPI according to the specified parameters in the SPI_InitTypeDef and create the associated handle.
Parameters	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• HAL: status

43.2.6 HAL_SPI_DeInit

Function Name	HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)
Function Description	DeInitializes the SPI peripheral.
Parameters	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• HAL: status

43.2.7 HAL_SPI_MspInit

Function Name	void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)
Function Description	SPI MSP Init.

Parameters	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None:

43.2.8 HAL_SPI_MspDeInit

Function Name	void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)
Function Description	SPI MSP DeInit.
Parameters	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None:

43.2.9 HAL_SPI_Transmit

Function Name	HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmit an amount of data in blocking mode.
Parameters	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent • Timeout: Timeout duration
Return values	• HAL: status

43.2.10 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	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent • Timeout: Timeout duration
Return values	• HAL: status

43.2.11 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	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pTxData: pointer to transmission data buffer • pRxData: pointer to reception data buffer to be • Size: amount of data to be sent

	• Timeout: Timeout duration
Return values	• HAL: status
43.2.12 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 no-blocking mode with Interrupt.
Parameters	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent
Return values	• HAL: status
43.2.13 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 no-blocking mode with Interrupt.
Parameters	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent
Return values	• HAL: status
43.2.14 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 no-blocking mode with Interrupt.
Parameters	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pTxData: pointer to transmission data buffer • pRxData: pointer to reception data buffer to be • Size: amount of data to be sent
Return values	• HAL: status
43.2.15 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 no-blocking mode with DMA.
Parameters	• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer

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

43.2.16 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 no-blocking mode with DMA.
- Parameters**
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
 - **pData:** pointer to data buffer
 - **Size:** amount of data to be sent
- Return values**
- **HAL:** status
- Notes**
- When the CRC feature is enabled the pData Length must be Size + 1.

43.2.17 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 no-blocking mode with DMA.
- Parameters**
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
 - **pTxData:** pointer to transmission data buffer
 - **pRxData:** pointer to reception data buffer
 - **Size:** amount of data to be sent
- Return values**
- **HAL:** status
- Notes**
- When the CRC feature is enabled the pRxData Length must be Size + 1

43.2.18 HAL_SPI_DMAPause

- Function Name** HAL_StatusTypeDef HAL_SPI_DMAPause
(SPI_HandleTypeDef * hspi)
- Function Description** Pauses the DMA Transfer.
- Parameters**
- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
- Return values**
- **HAL:** status

43.2.19 HAL_SPI_DMAResume

- Function Name** HAL_StatusTypeDef HAL_SPI_DMAResume
(SPI_HandleTypeDef * hspi)
- Function Description** Resumes the DMA Transfer.

Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	HAL: status

43.2.20 HAL_SPI_DMAStop

Function Name	HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)
Function Description	Stops the DMA Transfer.
Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	HAL: status

43.2.21 HAL_SPI_IRQHandler

Function Name	void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)
Function Description	This function handles SPI interrupt request.
Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	HAL: status

43.2.22 HAL_SPI_TxCpltCallback

Function Name	void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx Transfer completed callbacks.
Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	None:

43.2.23 HAL_SPI_RxCpltCallback

Function Name	void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Rx Transfer completed callbacks.
Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	None:

43.2.24 HAL_SPI_TxRxCpltCallback

Function Name	void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	None:

43.2.25 HAL_SPI_TxHalfCpltCallback

Function Name	void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx Half Transfer completed callbacks.
Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	None:

43.2.26 HAL_SPI_RxHalfCpltCallback

Function Name	void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Rx Half Transfer completed callbacks.
Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	None:

43.2.27 HAL_SPI_TxRxHalfCpltCallback

Function Name	void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	None:

43.2.28 HAL_SPI_ErrorCallback

Function Name	void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)
Function Description	SPI error callbacks.
Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	None:

43.2.29 HAL_SPI_GetState

Function Name	HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)
Function Description	Return the SPI state.
Parameters	hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	SPI: state

43.2.30 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

43.3 SPI Firmware driver defines

43.3.1 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

SPI Clock Phase

SPI_PHASE_1EDGE
 SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW
 SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE
 SPI_CRCCALCULATION_ENABLE

SPI data size

SPI_DATASIZE_8BIT
 SPI_DATASIZE_16BIT

SPI Direction mode

SPI_DIRECTION_2LINES
 SPI_DIRECTION_2LINES_RXONLY
 SPI_DIRECTION_1LINE

SPI Error Code

HAL_SPI_ERROR_NONE	No error
HAL_SPI_ERROR_MODF	MODF error
HAL_SPI_ERROR_CRC	CRC error

HAL_SPI_ERROR_OVR	OVR error
HAL_SPI_ERROR_FRE	FRE error
HAL_SPI_ERROR_DMA	DMA transfer error
HAL_SPI_ERROR_FLAG	Flag: RXNE, TXE, BSY

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE **Description:**

- Reset SPI handle state.

Parameters:

- **__HANDLE__**: specifies the SPI handle.
This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT

Description:

- Enable the specified SPI interrupts.

Parameters:

- **__HANDLE__**: specifies the SPI handle.
This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.
- **__INTERRUPT__**: specifies the interrupt source to enable. This parameter can be one of the following values:
 - SPI_IT_TXE: Tx buffer empty interrupt enable
 - SPI_IT_RXNE: RX buffer not empty interrupt enable
 - SPI_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

- **__HANDLE__**: specifies the SPI handle.
This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.
- **__INTERRUPT__**: specifies the interrupt source to disable. This parameter can be one of the following values:
 - SPI_IT_TXE: Tx buffer empty interrupt enable
 - SPI_IT_RXNE: RX buffer not empty interrupt enable
 - SPI_IT_ERR: Error interrupt enable

__HAL_SPI_GET_IT_SOURCE**Return value:**

- None

Description:

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

Parameters:

- **__HANDLE__**: specifies the SPI handle.
This parameter can be SPIx where x: 1 or 2 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).

Description:

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

Parameters:

- **__HANDLE__**: specifies the SPI handle.
This parameter can be SPIx where x: 1 or 2 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

Return value:

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

__HAL_SPI_CLEAR_CRCERRFLAG**Description:**

- Clear the SPI CRCERR pending flag.

Parameters:

	• <code>__HANDLE__</code>: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral. Return value: • None Description: • Clear the SPI MODF pending flag. Parameters: • <code>__HANDLE__</code>: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral. Return value: • None
<code>__HAL_SPI_CLEAR_MODFFLAG</code>	Description: • Clear the SPI MODF pending flag. Parameters: • <code>__HANDLE__</code>: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral. Return value: • None
<code>__HAL_SPI_CLEAR_OVRFLAG</code>	Description: • Clear the SPI OVR pending flag. Parameters: • <code>__HANDLE__</code>: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral. Return value: • None
<code>__HAL_SPI_CLEAR_FREFLAG</code>	Description: • Clear the SPI FRE pending flag. Parameters: • <code>__HANDLE__</code>: specifies the SPI handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral. Return value: • None
<code>__HAL_SPI_ENABLE</code>	Description: • Enables the SPI. Parameters: • <code>__HANDLE__</code>: specifies the SPI Handle. This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral. Return value: • None
<code>__HAL_SPI_DISABLE</code>	Description: • Disables the SPI.

Parameters:

- `__HANDLE__`: specifies the SPI Handle.
This parameter can be SPIx where x: 1 or 2 to select the SPI peripheral.

Return value:

- None

SPI Flag definition`SPI_FLAG_RXNE``SPI_FLAG_TXE``SPI_FLAG_CRCERR``SPI_FLAG_MODF``SPI_FLAG_OVR``SPI_FLAG_BSY``SPI_FLAG_FRE`***SPI Interrupt configuration definition***`SPI_IT_TXE``SPI_IT_RXNE``SPI_IT_ERR`***SPI mode***`SPI_MODE_SLAVE``SPI_MODE_MASTER`***SPI MSB LSB transmission***`SPI_FIRSTBIT_MSB``SPI_FIRSTBIT_LSB`***SPI Slave Select management***`SPI_NSS_SOFT``SPI_NSS_HARD_INPUT``SPI_NSS_HARD_OUTPUT`***SPI TI mode***`SPI_TIMODE_DISABLE``SPI_TIMODE_ENABLE`

44 HAL TIM Generic Driver

44.1 TIM Firmware driver registers structures

44.1.1 TIM_Base_InitTypeDef

Data Fields

- *uint32_t Prescaler*
- *uint32_t CounterMode*
- *uint32_t Period*
- *uint32_t ClockDivision*

Field Documentation

- ***uint32_t TIM_Base_InitTypeDef::Prescaler***
Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_Base_InitTypeDef::CounterMode***
Specifies the counter mode. This parameter can be a value of [TIM_Counter_Mode](#)
- ***uint32_t TIM_Base_InitTypeDef::Period***
Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::ClockDivision***
Specifies the clock division. This parameter can be a value of [TIM_ClockDivision](#)

44.1.2 TIM_OC_InitTypeDef

Data Fields

- *uint32_t OCMODE*
- *uint32_t Pulse*
- *uint32_t OCPolarity*
- *uint32_t OCFastMode*

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode***
Specifies the TIM mode. This parameter can be a value of [TIM_Output_Compare_and_PWM_modes](#)
- ***uint32_t TIM_OC_InitTypeDef::Pulse***
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OC_InitTypeDef::OCPolarity***
Specifies the output polarity. This parameter can be a value of [TIM_Output_Compare_Polarity](#)

- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***
Specifies the Fast mode state. This parameter can be a value of [TIM_Output_Fast_State](#)
Note: This parameter is valid only in PWM1 and PWM2 mode.

44.1.3 TIM_OnePulse_InitTypeDef

Data Fields

- ***uint32_t OCMODE***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFILTER***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***
Specifies the TIM mode. This parameter can be a value of [TIM_Output_Compare_and_PWM_modes](#)
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse***
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OnePulse_InitTypeDef::OCPolarity***
Specifies the output polarity. This parameter can be a value of [TIM_Output_Compare_Polarity](#)
- ***uint32_t TIM_OnePulse_InitTypeDef::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

44.1.4 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

44.1.5 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

44.1.6 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

44.1.7 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

44.1.8 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

44.1.9 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 [DMA_Handle_index](#)
- *HAL_LockTypeDef TIM_HandleTypeDef::Lock*
Locking object

- `__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State`
TIM operation state

44.2 TIM Firmware driver API description

44.2.1 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
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

44.2.2 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: HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_Base_Start_IT(), HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_OC_Start_IT(), HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_IC_Start_IT(), HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_TIM_PWM_Start_IT(), HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_Start_IT(), HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_DMA() or HAL_TIM_Encoder_Start_IT()
 6. The DMA Burst is managed with the two following functions:
HAL_TIM_DMABurst_WriteStart HAL_TIM_DMABurst_ReadStart

44.2.3 Timer Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Timer Base.
- Stop the Timer Base.
- Start the Timer Base and enable interrupt.
- Stop the Timer Base and disable interrupt.
- Start the Timer Base and enable DMA transfer.
- Stop the Timer 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\(\)*](#)

44.2.4 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_TIM_Base_GetState\(\)*](#)
- [*HAL_TIM_OC_GetState\(\)*](#)
- [*HAL_TIM_PWM_GetState\(\)*](#)
- [*HAL_TIM_IC_GetState\(\)*](#)
- [*HAL_TIM_OnePulse_GetState\(\)*](#)
- [*HAL_TIM_Encoder_GetState\(\)*](#)
- [*TIM_DMAError\(\)*](#)
- [*TIM_DMADelayPulseCplt\(\)*](#)
- [*TIM_DMACaptureCplt\(\)*](#)

44.2.5 HAL_TIM_Base_Init

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• htim: : TIM handle
Return values	• HAL: status

44.2.6 HAL_TIM_Base_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM Base peripheral.
Parameters	• htim: : TIM handle
Return values	• HAL: status

44.2.7 HAL_TIM_Base_MspInit

Function Name	void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Base MSP.
Parameters	• htim: : TIM handle
Return values	• None:

44.2.8 HAL_TIM_Base_MspDeInit

Function Name	void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Base MSP.
Parameters	• htim: : TIM handle
Return values	• None:

44.2.9 HAL_TIM_Base_Start

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)
Function Description	Starts the TIM Base generation.
Parameters	• htim: : TIM handle
Return values	• HAL: status

44.2.10 HAL_TIM_Base_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)
Function Description	Stops the TIM Base generation.
Parameters	• htim: : TIM handle

Return values • **HAL:** status

44.2.11 HAL_TIM_Base_Start_IT

Function Name **HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)**

Function Description Starts the TIM Base generation in interrupt mode.

Parameters • **htim:** : TIM handle

Return values • **HAL:** status

44.2.12 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

44.2.13 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

44.2.14 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

44.2.15 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

44.2.16 HAL_TIM_OC_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM peripheral.
Parameters	• htim: TIM Output Compare handle
Return values	• HAL: status

44.2.17 HAL_TIM_OC_MspInit

Function Name	void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Output Compare MSP.
Parameters	• htim: : TIM handle
Return values	• None:

44.2.18 HAL_TIM_OC_MspDeInit

Function Name	void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Output Compare MSP.
Parameters	• htim: : TIM handle
Return values	• None:

44.2.19 HAL_TIM_OC_Start

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Output Compare signal generation.
Parameters	• htim: : 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	• HAL: status

44.2.20 HAL_TIM_OC_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation.
Parameters	• htim: : 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

- **HAL:** status

44.2.21 HAL_TIM_OC_Start_IT

Function Name **HAL_StatusTypeDef HAL_TIM_OC_Start_IT**
(TIM_HandleTypeDef * htim, uint32_t Channel)

Function Description Starts the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** : 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

- **HAL:** status

44.2.22 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 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

44.2.23 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 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

44.2.24 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 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
44.2.25 HAL_TIM_PWM_Init	
Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• htim: : TIM handle
Return values	• HAL: status
44.2.26 HAL_TIM_PWM_DeInit	
Function Name	HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM peripheral.
Parameters	• htim: : TIM handle
Return values	• HAL: status
44.2.27 HAL_TIM_PWM_MspInit	
Function Name	void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM PWM MSP.
Parameters	• htim: : TIM handle
Return values	• None:
44.2.28 HAL_TIM_PWM_MspDeInit	
Function Name	void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM PWM MSP.
Parameters	• htim: : TIM handle
Return values	• None:
44.2.29 HAL_TIM_PWM_Start	
Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start

(TIM_HandleTypeDef * htim, uint32_t Channel)

Function Description	Starts the PWM signal generation.
Parameters	• htim: : 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	• HAL: status

44.2.30 HAL_TIM_PWM_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation.
Parameters	• htim: : 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	• HAL: status

44.2.31 HAL_TIM_PWM_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation in interrupt mode.
Parameters	• htim: : 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	• HAL: status

44.2.32 HAL_TIM_PWM_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation in interrupt mode.
Parameters	• htim: : 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	• HAL: status

44.2.33 HAL_TIM_PWM_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM PWM signal generation in DMA mode.
Parameters	• htim: : TIM handle • Channel: TIM Channels to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected • pData: The source Buffer address. This buffer contains the values which will be loaded inside the capture/compare registers. • Length: The length of data to be transferred from memory to TIM peripheral
Return values	• HAL: status

44.2.34 HAL_TIM_PWM_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM PWM signal generation in DMA mode.
Parameters	• htim: : TIM handle • Channel: TIM Channels to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL: status

44.2.35 HAL_TIM_IC_Init

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• htim: TIM Input Capture handle
Return values	• HAL: status

44.2.36 HAL_TIM_IC_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM peripheral.
Parameters	• htim: TIM Input Capture handle

Return values

- **HAL:** status

44.2.37 HAL_TIM_IC_MspInit

Function Name **void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)**

Function Description Initializes the TIM INput Capture MSP.

Parameters

- **htim:** : TIM handle

Return values

- **None:**

44.2.38 HAL_TIM_IC_MspDeInit

Function Name **void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)**

Function Description DeInitializes TIM Input Capture MSP.

Parameters

- **htim:** : TIM handle

Return values

- **None:**

44.2.39 HAL_TIM_IC_Start

Function Name **HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function Description Starts the TIM Input Capture measurement.

Parameters

- **htim:** : 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

- **HAL:** status

44.2.40 HAL_TIM_IC_Stop

Function Name **HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function Description Stops the TIM Input Capture measurement.

Parameters

- **htim:** : 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

- **HAL:** status

44.2.41 HAL_TIM_IC_Start_IT

Function Name **HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function Description Starts the TIM Input Capture measurement in interrupt mode.

Parameters	• htim: : 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	• HAL: status

44.2.42 HAL_TIM_IC_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Input Capture measurement in interrupt mode.
Parameters	• htim: : 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	• HAL: status

44.2.43 HAL_TIM_IC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM Input Capture measurement on in DMA mode.
Parameters	• htim: : 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 • pData: The destination Buffer address. • Length: The length of data to be transferred from TIM peripheral to memory.
Return values	• HAL: status

44.2.44 HAL_TIM_IC_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Input Capture measurement on in DMA mode.
Parameters	• htim: : 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

- **HAL:** status

44.2.45 HAL_TIM_OnePulse_Init

Function Name **HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)**

Function Description Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.

Parameters

- **htim:** TIM OnePulse handle
- **OnePulseMode:** Select the One pulse mode. This parameter can be one of the following values: TIM_OPMODE_SINGLE: Only one pulse will be generated. TIM_OPMODE_REPETITIVE: Repetitive pulses will be generated.

Return values

- **HAL:** status

44.2.46 HAL_TIM_OnePulse_DeInit

Function Name **HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)**

Function Description DeInitializes the TIM One Pulse.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **HAL:** status

44.2.47 HAL_TIM_OnePulse_MspInit

Function Name **void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)**

Function Description Initializes the TIM One Pulse MSP.

Parameters

- **htim:** : TIM handle

Return values

- **None:**

44.2.48 HAL_TIM_OnePulse_MspDeInit

Function Name **void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)**

Function Description DeInitializes TIM One Pulse MSP.

Parameters

- **htim:** : TIM handle

Return values

- **None:**

44.2.49 HAL_TIM_OnePulse_Start

Function Name **HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function Description Starts the TIM One Pulse signal generation.

Parameters

- **htim:** : TIM handle
- **OutputChannel:** : TIM Channels to be enabled. This

parameter is not used since both channels TIM_CHANNEL_1 and TIM_CHANNEL_2 are automatically selected.

Return values

- **HAL:** status

44.2.50 HAL_TIM_OnePulse_Stop

Function Name **HAL_StatusTypeDef HAL_TIM_OnePulse_Stop**
(TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function Description Stops the TIM One Pulse signal generation.

Parameters

- **htim:** : TIM handle
- **OutputChannel:** : TIM Channels to be disable. 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

44.2.51 HAL_TIM_OnePulse_Start_IT

Function Name **HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT**
(TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function Description Starts the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** : TIM 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

- **HAL:** status

44.2.52 HAL_TIM_OnePulse_Stop_IT

Function Name **HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT**
(TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function Description Stops the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** : TIM 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

- **HAL:** status

44.2.53 HAL_TIM_Encoder_Init

Function Name **HAL_StatusTypeDef HAL_TIM_Encoder_Init**
(TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)

Function Description Initializes the TIM Encoder Interface and create the associated handle.

Parameters

- **htim:** TIM Encoder Interface handle

- **sConfig:** TIM Encoder Interface configuration structure
- Return values
 - **HAL:** status

44.2.54 HAL_TIM_Encoder_DeInit

- Function Name **HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)**
- Function Description DeInitializes the TIM Encoder interface.
- Parameters
 - **htim:** TIM Encoder handle
- Return values
 - **HAL:** status

44.2.55 HAL_TIM_Encoder_MspInit

- Function Name **void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)**
- Function Description Initializes the TIM Encoder Interface MSP.
- Parameters
 - **htim:** : TIM handle
- Return values
 - **None:**

44.2.56 HAL_TIM_Encoder_MspDeInit

- Function Name **void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)**
- Function Description DeInitializes TIM Encoder Interface MSP.
- Parameters
 - **htim:** : TIM handle
- Return values
 - **None:**

44.2.57 HAL_TIM_Encoder_Start

- Function Name **HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
- Function Description Starts the TIM Encoder Interface.
- Parameters
 - **htim:** : 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_ALL: TIM Channel 1 and TIM Channel 2 are selected
- Return values
 - **HAL:** status

44.2.58 HAL_TIM_Encoder_Stop

- Function Name **HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
- Function Description Stops the TIM Encoder Interface.
- Parameters
 - **htim:** : 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_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

44.2.59 HAL_TIM_Encoder_Start_IT

Function Name HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function Description Starts the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** : 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_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

44.2.60 HAL_TIM_Encoder_Stop_IT

Function Name HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function Description Stops the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** : 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_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

44.2.61 HAL_TIM_Encoder_Start_DMA

Function Name HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)

Function Description Starts the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** : 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_ALL : TIM Channel 1 and 2 selected
- **pData1:** The destination Buffer address for IC1.
- **pData2:** The destination Buffer address for IC2.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

44.2.62 HAL_TIM_Encoder_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Encoder Interface in DMA mode.
Parameters	• htim: 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_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• HAL: status

44.2.63 HAL_TIM_IRQHandler

Function Name	void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)
Function Description	This function handles TIM interrupts requests.
Parameters	• htim: TIM handle
Return values	• None:

44.2.64 HAL_TIM_OC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)
Function Description	Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	• htim: TIM 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	• HAL: status

44.2.65 HAL_TIM_IC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig, uint32_t Channel)
Function Description	Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.
Parameters	• htim: TIM 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

44.2.66 HAL_TIM_PWM_ConfigChannel

Function Name HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function Description Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** : TIM handle
- **sConfig:** TIM PWM configuration structure
- **Channel:** TIM Channels to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

44.2.67 HAL_TIM_OnePulse_ConfigChannel

Function Name HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)

Function Description Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.

Parameters

- **htim:** : TIM 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

44.2.68 HAL_TIM_DMABurst_WriteStart

Function Name HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)

Function Description Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

tion
Description

Parameters

- **htim:** : TIM handle
- **BurstBaseAddress:** TIM Base address from when the DMA will starts the Data write. This parameters can be on of the following values:
TIM_DMABASE_CR1TIM_DMABASE_CR2TIM_DMABASE_SMCRTIM_DMABASE_DIERTIM_DMABASE_SRTIM_DMABASE_EGRTIM_DMABASE_CCMR1TIM_DMABASE_CCMR2TIM_DMABASE_CCERTIM_DMABASE_CNTTIM_DMABASE_PSCTIM_DMABASE_ARRTIM_DMABASE_CCR1TIM_DMABASE_CCR2TIM_DMABASE_CCR3TIM_DMABASE_CCR4TIM_DMABASE_DCR
- **BurstRequestSrc:** TIM DMA Request sources. This parameters can be on 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_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

44.2.69 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

44.2.70 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 Name

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 when the DMA will starts the Data read. This parameters can be on of the following values: TIM_DMABASE_CR1TIM_DMABASE_CR2TIM_DMABASE_SMCRTIM_DMABASE_DIERTIM_DMABASE_SRTIM_DMABASE_EGRTIM_DMABASE_CCMR1TIM_DMABASE_CCMR2TIM_DMABASE_CCERTIM_DMABASE_CNTTIM_DMABASE_PSCTIM_DMABASE_ARRTIM_DMABASE_CCR1TIM_DMABASE_CCR2TIM_DMABASE_CCR3TIM_DMABASE_CCR4TIM_DMABASE_DCR • BurstRequestSrc: TIM DMA Request sources. This parameters can be on 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_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

44.2.71 HAL_TIM_DMABurst_ReadStop

Function Name	HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)
Function Description	Stop the DMA burst reading.
Parameters	• htim: : TIM handle • BurstRequestSrc: TIM DMA Request sources to disable.
Return values	• HAL: status

44.2.72 HAL_TIM_GenerateEvent

Function Name	HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)
Function Description	Generate a software event.
Parameters	• htim: : TIM handle • EventSource: specifies the event source. This parameter

can be one of the following values:

TIM_EventSource_Update: Timer update Event
 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_TRIGGER : Timer Trigger Event source

Return values

- **HAL:** status

Notes

- TIM6 can only generate an update event.

44.2.73 HAL_TIM_ConfigOCrefClear

Function Name

HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear
(TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)

Function Description

Configures the OCRef clear feature.

Parameters

- **htim:** : TIM handle
- **sClearInputConfig:** pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel:** specifies the TIM Channel. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected
TIM_CHANNEL_2: TIM Channel 2 selected
TIM_CHANNEL_3: TIM Channel 3 selected
TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

44.2.74 HAL_TIM_ConfigClockSource

Function Name

HAL_StatusTypeDef HAL_TIM_ConfigClockSource
(TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)

Function Description

Configures the clock source to be used.

Parameters

- **htim:** : TIM handle
- **sClockSourceConfig:** pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.

Return values

- **HAL:** status

44.2.75 HAL_TIM_ConfigTI1Input

Function Name

HAL_StatusTypeDef HAL_TIM_ConfigTI1Input
(TIM_HandleTypeDef * htim, uint32_t TI1_Selection)

Function Description

Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.

Parameters

- **htim:** : TIM handle

- **TI1_Selection:** Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values: TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 inputTIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

Return values

- **HAL:** status

44.2.76 HAL_TIM_SlaveConfigSynchronization

Function Name **HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)**

Function Description Configures the TIM in Slave mode.

Parameters

- **htim:** : TIM handle
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values

- **HAL:** status

44.2.77 HAL_TIM_SlaveConfigSynchronization_IT

Function Name **HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)**

Function Description Configures the TIM in Slave mode in interrupt mode.

Parameters

- **htim:** : TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values

- **HAL:** status

44.2.78 HAL_TIM_ReadCapturedValue

Function Name **uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function Description Read the captured value from Capture Compare unit.

Parameters

- **htim:** : TIM handle
- **Channel:** TIM Channels to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **Captured:** value

44.2.79 HAL_TIM_PeriodElapsedCallback

Function Name **void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)**

Function Description Period elapsed callback in non blocking mode.

Parameters

- **htim:** : TIM handle

Return values

- **None:**

44.2.80 HAL_TIM_OC_DelayElapsedCallback

Function Name **void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)**

Function Description Output Compare callback in non blocking mode.

Parameters

- **htim:** : TIM handle

Return values

- **None:**

44.2.81 HAL_TIM_IC_CaptureCallback

Function Name **void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)**

Function Description Input Capture callback in non blocking mode.

Parameters

- **htim:** TIM IC handle

Return values

- **None:**

44.2.82 HAL_TIM_PWM_PulseFinishedCallback

Function Name **void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)**

Function Description PWM Pulse finished callback in non blocking mode.

Parameters

- **htim:** : TIM handle

Return values

- **None:**

44.2.83 HAL_TIM_TriggerCallback

Function Name **void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)**

Function Description Hall Trigger detection callback in non blocking mode.

Parameters

- **htim:** : TIM handle

Return values

- **None:**

44.2.84 HAL_TIM_ErrorCallback

Function Name **void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)**

Function Description Timer error callback in non blocking mode.

Parameters	• htim: : TIM handle
Return values	• None:

44.2.85 HAL_TIM_Base_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Base state.
Parameters	• htim: : TIM handle
Return values	• HAL: state

44.2.86 HAL_TIM_OC_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM OC state.
Parameters	• htim: TIM Output Compare handle
Return values	• HAL: state

44.2.87 HAL_TIM_PWM_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM PWM state.
Parameters	• htim: : TIM handle
Return values	• HAL: state

44.2.88 HAL_TIM_IC_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Input Capture state.
Parameters	• htim: : TIM handle
Return values	• HAL: state

44.2.89 HAL_TIM_OnePulse_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM One Pulse Mode state.
Parameters	• htim: TIM OPM handle
Return values	• HAL: state

44.2.90 HAL_TIM_Encoder_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Encoder Mode state.
Parameters	• htim: : TIM handle
Return values	• HAL: state
44.2.91 TIM_DMAError	
Function Name	void TIM_DMAError (DMA_HandleTypeDef * hdma)
Function Description	TIM DMA error callback.
Parameters	• hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• None:
44.2.92 TIM_DMADelayPulseCplt	
Function Name	void TIM_DMADelayPulseCplt (DMA_HandleTypeDef * hdma)
Function Description	TIM DMA Delay Pulse complete callback.
Parameters	• hdma: : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• None:
44.2.93 TIM_DMACaptureCplt	
Function Name	void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)
Function Description	TIM DMA Capture complete callback.
Parameters	• hdma: : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• None:
44.3 TIM Firmware driver defines	
44.3.1 TIM	
<i>TIM channels</i>	
TIM_CHANNEL_1	
TIM_CHANNEL_2	
TIM_CHANNEL_3	
TIM_CHANNEL_4	
TIM_CHANNEL_ALL	
<i>Clear input polarity</i>	

TIM_CLEARINPUTPOLARITY_INVERTED	Polarity for ETRx pin
TIM_CLEARINPUTPOLARITY_NONINVERTED	Polarity for ETRx pin

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.

Clear input source

TIM_CLEARINPUTSOURCE_ETR
TIM_CLEARINPUTSOURCE_NONE

Clock division

TIM_CLOCKDIVISION_DIV1
TIM_CLOCKDIVISION_DIV2
TIM_CLOCKDIVISION_DIV4

Clock polarity

TIM_CLOCKPOLARITY_INVERTED	Polarity for ETRx clock sources
TIM_CLOCKPOLARITY_NONINVERTED	Polarity for ETRx clock sources
TIM_CLOCKPOLARITY_RISING	Polarity for Tlx clock sources
TIM_CLOCKPOLARITY_FALLING	Polarity for Tlx clock sources
TIM_CLOCKPOLARITY_BOTHEDGE	Polarity for Tlx clock sources

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.

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

Counter mode

TIM_COUNTERMODE_UP
TIM_COUNTERMODE_DOWN
TIM_COUNTERMODE_CENTERALIGNED1
TIM_COUNTERMODE_CENTERALIGNED2
TIM_COUNTERMODE_CENTERALIGNED3

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_CCR1
TIM_DMABASE_CCR2
TIM_DMABASE_CCR3
TIM_DMABASE_CCR4
TIM_DMABASE_DCR
TIM_DMABASE_OR

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

DMA sources

TIM_DMA_UPDATE
TIM_DMA_CC1
TIM_DMA_CC2
TIM_DMA_CC3
TIM_DMA_CC4
TIM_DMA_TRIGGER

Encoder Mode

TIM_ENCODERMODE_TI1
TIM_ENCODERMODE_TI2
TIM_ENCODERMODE_TI12

ETR polarity

TIM_ETRPOLARITY_INVERTED Polarity for ETR source
TIM_ETRPOLARITY_NONINVERTED Polarity for ETR source

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

Event sources

TIM_EVENTSOURCE_UPDATE
TIM_EVENTSOURCE_CC1
TIM_EVENTSOURCE_CC2
TIM_EVENTSOURCE_CC3

TIM_EVENTSOURCE_CC4

TIM_EVENTSOURCE_TRIGGER

TIM Exported Constants

IS_TIM_PERIOD

IS_TIM_PRESCALER

IS_TIM_COUNTER_MODE

IS_TIM_CLOCKDIVISION_DIV

IS_TIM_PWM_MODE

IS_TIM_OC_MODE

IS_TIM_FAST_STATE

IS_TIM_OC_POLARITY

IS_TIM_CHANNELS

IS_TIM_OPM_CHANNELS

IS_TIM_IC_POLARITY

IS_TIM_IC_PRESCALER

IS_TIM_OPM_MODE

IS_TIM_ENCODER_MODE

IS_TIM_DMA_SOURCE

IS_TIM_EVENT_SOURCE

IS_TIM_CLOCKSOURCE

IS_TIM_CLOCKPOLARITY

IS_TIM_CLOCKPRESCALER

IS_TIM_CLOCKFILTER

IS_TIM_CLEARINPUT_SOURCE

IS_TIM_CLEARINPUT_POLARITY

IS_TIM_CLEARINPUT_PRESCALER

IS_TIM_CLEARINPUT_FILTER

IS_TIM_TRGO_SOURCE

IS_TIM_SLAVE_MODE

IS_TIM_MSM_STATE

IS_TIM_TRIGGER_SELECTION

IS_TIM_INTERNAL_TRIGGEREVENT_SELECTION

IS_TIM_TRIGGERPOLARITY

IS_TIM_TRIGGERPRESCALER

IS_TIM_TRIGGERFILTER

IS_TIM_TI1SELECTION

IS_TIM_DMA_BASE

IS_TIM_DMA_LENGTH

IS_TIM_IC_FILTER

TIM Exported Macro

__HAL_TIM_RESET_HANDLE_STATE

Description:

- Reset UART handle state.

Parameters:

- __HANDLE__: : TIM handle

Return value:

- None

__HAL_TIM_ENABLE

Description:

- Enable the TIM peripheral.

Parameters:

- __HANDLE__: : TIM handle

Return value:

- None

TIM_CCER_CCxE_MASK

__HAL_TIM_DISABLE

Description:

- Disable the TIM peripheral.

Parameters:

- __HANDLE__: : TIM handle

Return value:

- None

__HAL_TIM_ENABLE_IT

__HAL_TIM_ENABLE_DMA

__HAL_TIM_DISABLE_IT

__HAL_TIM_DISABLE_DMA

__HAL_TIM_GET_FLAG

__HAL_TIM_CLEAR_FLAG

__HAL_TIM_GET_IT_SOURCE

__HAL_TIM_CLEAR_IT

__HAL_TIM_IS_TIM_COUNTING_DOWN

__HAL_TIM_SET_PRESCALER

TIM_SET_ICPRESCALERVALUE

TIM_RESET_ICPRESCALERVALUE

TIM_SET_CAPTUREPOLARITY
TIM_RESET_CAPTUREPOLARITY

__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:

	• <code>__HANDLE__</code>: : TIM handle. • <code>__COUNTER__</code>: specifies the Counter register new value.
	Return value: • None
<code>__HAL_TIM_GET_COUNTER</code>	Description: • Gets the TIM Counter Register value on runtime.
	Parameters: • <code>__HANDLE__</code>: : TIM handle.
	Return value: • None
<code>__HAL_TIM_SET_AUTORELOAD</code>	Description: • Sets the TIM Autoreload Register value on runtime without calling another time any Init function.
	Parameters: • <code>__HANDLE__</code>: : TIM handle. • <code>__AUTORELOAD__</code>: specifies the Counter register new value.
	Return value: • None
<code>__HAL_TIM_GET_AUTORELOAD</code>	Description: • Gets the TIM Autoreload Register value on runtime.
	Parameters: • <code>__HANDLE__</code>: : TIM handle.
	Return value: • None
<code>__HAL_TIM_SET_CLOCKDIVISION</code>	Description: • Sets the TIM Clock Division value on runtime without calling another time any Init function.
<code>N</code>	Parameters: • <code>__HANDLE__</code>: : TIM handle. • <code>__CKD__</code>: specifies the clock division value. This parameter can be one of the following value: – <code>TIM_CLOCKDIVISION_DIV1</code> – <code>TIM_CLOCKDIVISION_DIV2</code> – <code>TIM_CLOCKDIVISION_DIV4</code>
	Return value:

`__HAL_TIM_GET_CLOCKDIVISION`

- None

Description:

- Gets the TIM Clock Division value on runtime.

Parameters:

- `__HANDLE__`: TIM handle.

Return value:

- None

`__HAL_TIM_SET_ICPRESCALER`

Description:

- Sets the TIM Input Capture prescaler on runtime without calling another time

Parameters:

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: TIM Channel 1 selected
 - `TIM_CHANNEL_2`: TIM Channel 2 selected
 - `TIM_CHANNEL_3`: TIM Channel 3 selected
 - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `__ICPSC__`: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
 - `TIM_ICPSC_DIV1`: no prescaler
 - `TIM_ICPSC_DIV2`: capture is done once every 2 events
 - `TIM_ICPSC_DIV4`: capture is done once every 4 events
 - `TIM_ICPSC_DIV8`: capture is done once every 8 events

Return value:

- None

`__HAL_TIM_GET_ICPRESCALER`

Description:

- Gets the TIM Input Capture prescaler on runtime.

Parameters:

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: get input capture 1 prescaler value
 - `TIM_CHANNEL_2`: get input capture 2

- prescaler value
- TIM_CHANNEL_3: get input capture 3 prescaler value
- TIM_CHANNEL_4: get input capture 4 prescaler value

Return value:

- None

__HAL_TIM_URS_ENABLE**Description:**

- Set the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS 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 URS 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_BOTHEDGE: Rising and Falling Edge

Return value:

- None

Notes:

- The polarity TIM_INPUTCHANNELPOLARITY_BOTHEDGE is not authorized for TIM Channel 4.

Flag definition

TIM_FLAG_UPDATE

TIM_FLAG_CC1

TIM_FLAG_CC2

TIM_FLAG_CC3

TIM_FLAG_CC4

TIM_FLAG_TRIGGER

TIM_FLAG_CC1OF

TIM_FLAG_CC2OF

TIM_FLAG_CC3OF

TIM_FLAG_CC4OF

Input capture polarity

TIM_ICPOLARITY_RISING

TIM_ICPOLARITY_FALLING

TIM_ICPOLARITY_BOTHEDGE

Input capture prescaler

TIM_ICPSC_DIV1 Capture performed each time an edge is detected on the capture input

TIM_ICPSC_DIV2 Capture performed once every 2 events

TIM_ICPSC_DIV4 Capture performed once every 4 events

TIM_ICPSC_DIV8 Capture performed once every 8 events

Input capture selection

TIM_ICSELECTION_DIRECTTI TIM Input 1, 2, 3 or 4 is selected to be connected to

IC1, IC2, IC3 or IC4, respectively

TIM_ICSELECTION_INDIRECTTI TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively

TIM_ICSELECTION_TRC TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

IS_TIM_IC_SELECTION

Input channel polarity

TIM_INPUTCHANNELPOLARITY_RISING Polarity for Tlx source

TIM_INPUTCHANNELPOLARITY_FALLING Polarity for Tlx source

TIM_INPUTCHANNELPOLARITY_BOTHEDGE Polarity for Tlx source

Interrupt definition

TIM_IT_UPDATE

TIM_IT_CC1

TIM_IT_CC2

TIM_IT_CC3

TIM_IT_CC4

TIM_IT_TRIGGER

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

Master slave mode

TIM_MASTERSLAVEMODE_ENABLE

TIM_MASTERSLAVEMODE_DISABLE

One pulse mode

TIM_OPMODE_SINGLE

TIM_OPMODE_REPETITIVE

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

Output compare N state

TIM_OUTPUTNSTATE_DISABLE

TIM_OUTPUTNSTATE_ENABLE

Output compare polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

Output compare state

TIM_OUTPUTSTATE_DISABLE

TIM_OUTPUTSTATE_ENABLE

Output fast state

TIM_OCFAST_DISABLE

TIM_OCFAST_ENABLE

Slave mode

TIM_SLAVEMODE_DISABLE

TIM_SLAVEMODE_RESET

TIM_SLAVEMODE_GATED

TIM_SLAVEMODE_TRIGGER

TIM_SLAVEMODE_EXTERNAL1

TI1 selection

TIM_TI1SELECTION_CH1

TIM_TI1SELECTION_XORCOMBINATION

Trigger polarity

TIM_TRIGGERPOLARITY_INVERTED Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_NONINVERTED Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_RISING Polarity for TlxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_FALLING Polarity for TlxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_BOTHEDGE Polarity for TlxFPx or TI1_ED trigger sources

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.

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

45 HAL TIM Extension Driver

45.1 TIMEx Firmware driver registers structures

45.1.1 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](#)

45.2 TIMEx Firmware driver API description

45.2.1 TIM specific features integration

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for: Input Capture Output Compare PWM generation (Edge and Center-aligned Mode) One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

45.2.2 How to use this driver

1. Enable the TIM interface clock using `__HAL_RCC_TIMx_CLK_ENABLE();`
2. 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 operating mode using one of the configuration function of this driver:

- HAL_TIMEx_MasterConfigSynchronization() to configure the peripheral in master mode.
5. Remap the Timer I/O using HAL_TIMEx_RemapConfig() API.

45.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure Master and the Slave synchronization.

This section contains the following APIs:

- [HAL_TIMEx_MasterConfigSynchronization\(\)](#)
- [HAL_TIMEx_RemapConfig\(\)](#)

45.2.4 HAL_TIMEx_MasterConfigSynchronization

Function Name	HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)
Function Description	Configures the TIM in master mode.
Parameters	• htim: TIM handle. • sMasterConfig: pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.
Return values	• HAL: status

45.2.5 HAL_TIMEx_RemapConfig

Function Name	HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)
Function Description	Configures the remapping of the TIM2, TIM3, TIM21 and TIM22 inputs.
Parameters	• htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • Remap: specifies the TIM input remapping source. This parameter is a combination of the following values depending on TIM instance:
Return values	• HAL: status
Notes	• It is not possible to connect TIM2 and TIM21 on GPIOB5_AF4 at the same time. When selecting TIM3_TI2_GPIOB5_AF4, Channel2 of TIM3 will be connected to GPIOB5_AF4 and Channel2 of TIM22 will be connected to some other GPIOs. (refer to alternate functions for more details) When selecting TIM3_TI2_GPIO_DEF, Channel2 of Timer 3 will be connected an GPIO (other than GPIOB5_AF4) and Channel2 of TIM22 will be connected to GPIOB5_AF4.

45.3 TIMEx Firmware driver defines

45.3.1 TIMEx

Remaping

TIM2_ETR_GPIO
TIM2_ETR_HSI48
TIM2_ETR_HSI16
TIM2_ETR_LSE
TIM2_ETR_COMP2_OUT
TIM2_ETR_COMP1_OUT
TIM2_TI4_GPIO
TIM2_TI4_COMP2
TIM2_TI4_COMP1
TIM21_ETR_GPIO
TIM21_ETR_COMP2_OUT
TIM21_ETR_COMP1_OUT
TIM21_ETR_LSE
TIM21_TI1_GPIO
TIM21_TI1_MCO
TIM21_TI1_RTC_WKUT_IT
TIM21_TI1_HSE_RTC
TIM21_TI1_MSI
TIM21_TI1_LSE
TIM21_TI1_LSI
TIM21_TI1_COMP1_OUT
TIM21_TI2_GPIO
TIM21_TI2_COMP2_OUT
TIM22_ETR_LSE
TIM22_ETR_COMP2_OUT
TIM22_ETR_COMP1_OUT
TIM22_ETR_GPIO
TIM22_TI1_GPIO1
TIM22_TI1_COMP2_OUT
TIM22_TI1_COMP1_OUT
TIM22_TI1_GPIO2
TIM3_TI4_GPIO_DEF
TIM3_TI4_GPIOC9_AF2

TIM3_TI2_GPIO_DEF
TIM3_TI2_GPIOB5_AF4
TIM3_TI1_USB_SOF
TIM3_TI1_GPIO
TIM3_ETR_GPIO
TIM3_ETR_HSI
IS_TIM_REMAP
IS_CHANNEL_AVAILABLE

Trigger selection

TIM_TRGO_RESET
TIM_TRGO_ENABLE
TIM_TRGO_UPDATE
TIM_TRGO_OC1
TIM_TRGO_OC1REF
TIM_TRGO_OC2REF
TIM_TRGO_OC3REF
TIM_TRGO_OC4REF
IS_TIM_TRGO_SOURCE

46 HAL TSC Generic Driver

46.1 TSC Firmware driver registers structures

46.1.1 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

46.1.2 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

46.1.3 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

46.2 TSC Firmware driver API description

46.2.1 TSC specific features

1. Proven and robust surface charge transfer acquisition principle
2. Supports up to 3 capacitive sensing channels per group
3. Capacitive sensing channels can be acquired in parallel offering a very good response time
4. Spread spectrum feature to improve system robustness in noisy environments
5. Full hardware management of the charge transfer acquisition sequence
6. Programmable charge transfer frequency
7. Programmable sampling capacitor I/O pin
8. Programmable channel I/O pin
9. Programmable max count value to avoid long acquisition when a channel is faulty
10. Dedicated end of acquisition and max count error flags with interrupt capability
11. One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
12. Compatible with proximity, touchkey, linear and rotary touch sensor implementation

46.2.2 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.

46.2.3 IO Operation functions

This section provides functions allowing to:

- Start acquisition in polling mode.
- Start acquisition in interrupt mode.
- Stop conversion in polling mode.
- Stop conversion in interrupt mode.
- Get group acquisition status.

- Get group acquisition value.

This section contains the following APIs:

- [HAL_TSC_Start\(\)](#)
- [HAL_TSC_Start_IT\(\)](#)
- [HAL_TSC_Stop\(\)](#)
- [HAL_TSC_Stop_IT\(\)](#)
- [HAL_TSC_GroupGetStatus\(\)](#)
- [HAL_TSC_GroupGetValue\(\)](#)

46.2.4 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\(\)](#)

46.2.5 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\(\)](#)
- [HAL_TSC_ConvCpltCallback\(\)](#)
- [HAL_TSC_ErrorCallback\(\)](#)

46.2.6 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_DeInit\(\)](#)
- [HAL_TSC_MspInit\(\)](#)
- [HAL_TSC_MspDeInit\(\)](#)

46.2.7 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
46.2.8	HAL_TSC_DeInit	
	Function Name	HAL_StatusTypeDef HAL_TSC_DeInit (TSC_HandleTypeDef * htsc)
	Function Description	Deinitializes the TSC peripheral registers to their default reset values.
	Parameters	• htsc: TSC handle
	Return values	• HAL: status
46.2.9	HAL_TSC_MspInit	
	Function Name	void HAL_TSC_MspInit (TSC_HandleTypeDef * htsc)
	Function Description	Initializes the TSC MSP.
	Parameters	• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
	Return values	• None:
46.2.10	HAL_TSC_MspDeInit	
	Function Name	void HAL_TSC_MspDeInit (TSC_HandleTypeDef * htsc)
	Function Description	DeInitializes the TSC MSP.
	Parameters	• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
	Return values	• None:
46.2.11	HAL_TSC_Start	
	Function Name	HAL_StatusTypeDef HAL_TSC_Start (TSC_HandleTypeDef * htsc)
	Function Description	Starts the acquisition.
	Parameters	• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
	Return values	• HAL: status
46.2.12	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	• htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
	Return values	• HAL: status.

46.2.13 HAL_TSC_Stop

Function Name	HAL_StatusTypeDef HAL_TSC_Stop (TSC_HandleTypeDef * htsc)
Function Description	Stops the acquisition previously launched in polling mode.
Parameters	htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	HAL: status

46.2.14 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	htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	HAL: status

46.2.15 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	htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. gx_index: Index of the group
Return values	Group: status

46.2.16 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	htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. gx_index: Index of the group
Return values	Acquisition: measure

46.2.17 HAL_TSC_IOConfig

Function Name	HAL_StatusTypeDef HAL_TSC_IOConfig (TSC_HandleTypeDef * htsc, TSC_IOConfigTypeDef * config)
Function Description	Configures TSC IOs.
Parameters	htsc: pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. config: pointer to the configuration structure.

Return values

- **HAL:** status

46.2.18 HAL_TSC_IODischarge

Function Name **HAL_StatusTypeDef HAL_TSC_IODischarge (TSC_HandleTypeDef * htsc, uint32_t choice)**

Function Description Discharge TSC IOs.

Parameters

- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
- **choice:** enable or disable

Return values

- **HAL:** status

46.2.19 HAL_TSC_GetState

Function Name **HAL_TSC_StateTypeDef HAL_TSC_GetState (TSC_HandleTypeDef * htsc)**

Function Description Return the TSC state.

Parameters

- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **HAL:** state

46.2.20 HAL_TSC_PollForAcquisition

Function Name **HAL_StatusTypeDef HAL_TSC_PollForAcquisition (TSC_HandleTypeDef * htsc)**

Function Description Start acquisition and wait until completion.

Parameters

- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **HAL:** state

Notes

- There is no need of a timeout parameter as the max count error is already managed by the TSC peripheral.

46.2.21 HAL_TSC_IRQHandler

Function Name **void HAL_TSC_IRQHandler (TSC_HandleTypeDef * htsc)**

Function Description Handles TSC interrupt request.

Parameters

- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **None:**

46.2.22 HAL_TSC_ConvCpltCallback

Function Name **void HAL_TSC_ConvCpltCallback (TSC_HandleTypeDef * htsc)**

Function Description Acquisition completed callback in non blocking mode.

Parameters

- **htsc:** pointer to a TSC_HandleTypeDef structure that

contains the configuration information for the specified TSC.

Return values

- **None:**

46.2.23 HAL_TSC_ErrorCallback

Function Name **void HAL_TSC_ErrorCallback (TSC_HandleTypeDef * htsc)**

Function Description Error callback in non blocking mode.

Parameters

- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **None:**

46.2.24 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

46.2.25 HAL_TSC_DeInit

Function Name **HAL_StatusTypeDef HAL_TSC_DeInit (TSC_HandleTypeDef * htsc)**

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

Parameters

- **htsc:** TSC handle

Return values

- **HAL:** status

46.2.26 HAL_TSC_MspInit

Function Name **void HAL_TSC_MspInit (TSC_HandleTypeDef * htsc)**

Function Description Initializes the TSC MSP.

Parameters

- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **None:**

46.2.27 HAL_TSC_MspDeInit

Function Name **void HAL_TSC_MspDeInit (TSC_HandleTypeDef * htsc)**

Function Description DeInitializes the TSC MSP.

Parameters

- **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **None:**

46.3 TSC Firmware driver defines

46.3.1 TSC

TSC Exported Constants

IS_TSC_ALL_INSTANCE
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_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
IS_TSC_SS
IS_TSC_SSD
TSC_SS_PRESC_DIV1
TSC_SS_PRESC_DIV2
IS_TSC_SS_PRESC
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_MCV_255
TSC_MCV_511
TSC_MCV_1023
TSC_MCV_2047
TSC_MCV_4095
TSC_MCV_8191
TSC_MCV_16383
IS_TSC_MCV
TSC_IODEF_OUT_PP_LOW
TSC_IODEF_IN_FLOAT
IS_TSC_IODEF
TSC_SYNC_POLARITY_FALLING
TSC_SYNC_POLARITY_RISING
IS_TSC_SYNC_POL
TSC_ACQ_MODE_NORMAL
TSC_ACQ_MODE_SYNCHRO
IS_TSC_ACQ_MODE
TSC_IOMODE_UNUSED
TSC_IOMODE_CHANNEL
TSC_IOMODE_SHIELD

TSC_IOMODE_SAMPLING
IS_TSC_IOMODE
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 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:

__HAL_TSC_DISABLE

- __HANDLE__: TSC handle

Return value:

- None

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

Description:

- Set synchronization polarity to rising edge and high level.

Parameters:

- `__HANDLE__`: TSC handle

Return value:

- None

Description:

- Enable TSC interrupt.

Parameters:

- `__HANDLE__`: TSC handle
- `__INTERRUPT__`: TSC interrupt

Return value:

- None

Description:

- Disable TSC interrupt.

Parameters:

- `__HANDLE__`: TSC handle
- `__INTERRUPT__`: TSC interrupt

Return value:

- None

Description:

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

Parameters:

- `__HANDLE__`: TSC Handle
- `__INTERRUPT__`: TSC interrupt

Return value:

- SET: or RESET

Description:

- Get the selected TSC's flag status.

Parameters:

- `__HANDLE__`: TSC handle

`__HAL_TSC_SET_SYNC_POL_RISE_HIGH`

`__HAL_TSC_ENABLE_IT`

`__HAL_TSC_DISABLE_IT`

`__HAL_TSC_GET_IT_SOURCE`

`__HAL_TSC_GET_FLAG`

`__HAL_TSC_CLEAR_FLAG`

- `__FLAG__`: TSC flag

Return value:

- SET: or RESET

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

__HAL_TSC_ENABLE_CHANNEL

Return value:

- None

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

Description:

- Disable a group of sampling IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None

__HAL_TSC_ENABLE_GROUP

Description:

- Enable acquisition groups.

Parameters:

- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:

__HAL_TSC_DISABLE_GROUP

- None

Description:

- Disable acquisition groups.

Parameters:

- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:

- None

__HAL_TSC_GET_GROUP_STATUS

Description:

- Gets acquisition group status.

Parameters:

- __HANDLE__: TSC Handle
- __GX_INDEX__: Group index

Return value:

- SET: or RESET

TSC Flags Definition

TSC_FLAG_EOA

TSC_FLAG_MCE

TSC Interrupts Definition

TSC_IT_EOA

TSC_IT_MCE

IS_TSC_MCE_IT

47 HAL UART Generic Driver

47.1 UART Firmware driver registers structures

47.1.1 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, Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate))) If oversampling is 8, Baud Rate Register[15:4] = ((2 * PCLKx) / ((huart->Init.BaudRate))) [15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * PCLKx) / ((huart->Init.BaudRate))) [3:0]) >> 1
- ***uint32_t UART_InitTypeDef::WordLength***
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [UARTEx_Word_Length](#)
- ***uint32_t UART_InitTypeDef::StopBits***
Specifies the number of stop bits transmitted. This parameter can be a value of [UART_Stop_Bits](#)
- ***uint32_t UART_InitTypeDef::Parity***
Specifies the parity mode. This parameter can be a value of [UART_Parity](#)
Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t UART_InitTypeDef::Mode***
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [UART_Mode](#)
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of [UART_Hardware_Flow_Control](#)
- ***uint32_t UART_InitTypeDef::OverSampling***
Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to fPCLK/8). This parameter can be a value of [UART_Over_Sampling](#)
- ***uint32_t UART_InitTypeDef::OneBitSampling***
Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of [UART_One_Bit](#)

47.1.2 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 [UARTEEx_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](#)

47.1.3 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_InitTypeDef UART_HandleTypeDef::Init*
- *UART_AdvFeatureInitTypeDef UART_HandleTypeDef::AdvancedInit*
- *uint8_t* UART_HandleTypeDef::pTxBuffPtr*
- *uint16_t UART_HandleTypeDef::TxXferSize*
- *uint16_t UART_HandleTypeDef::TxXferCount*
- *uint8_t* UART_HandleTypeDef::pRxBuffPtr*
- *uint16_t UART_HandleTypeDef::RxXferSize*
- *uint16_t UART_HandleTypeDef::RxXferCount*
- *uint16_t UART_HandleTypeDef::Mask*
- *DMA_HandleTypeDef* UART_HandleTypeDef::hdmatx*
- *DMA_HandleTypeDef* UART_HandleTypeDef::hdmarx*
- *HAL_LockTypeDef UART_HandleTypeDef::Lock*
- *__IO HAL_UART_StateTypeDef UART_HandleTypeDef::State*
- *__IO uint32_t UART_HandleTypeDef::ErrorCode*

47.2 UART Firmware driver API description

47.2.1 Initialization and Configuration functions

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

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Stop Bit
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), the possible UART frame formats are as listed in the following table.
 - Hardware flow control

- Receiver/transmitter modes
- Over Sampling Method
- One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
 - TX and/or RX pin level inversion
 - data logical level inversion
 - RX and TX pins swap
 - RX overrun detection disabling
 - DMA disabling on RX error
 - MSB first on communication line
 - auto Baud rate detection

Table 21: UART frame formats (2 M bits)

M1 M0 bits	PCE bit	UART frame
00	0	SB 8 bit data STB
00	1	SB 7 bit data PB STB
01	0	SB 9 bit data STB
01	1	SB 8 bit data PB STB
10	0	SB 7 bit data STB
10	1	SB 6 bit data PB STB

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init() and HAL_MultiProcessorEx_Init() API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and UART multiprocessor mode 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_LIN_Init\(\)*](#)
- [*HAL_MultiProcessor_Init\(\)*](#)
- [*HAL_UART_DeInit\(\)*](#)
- [*HAL_UART_MspInit\(\)*](#)
- [*HAL_UART_MspDeInit\(\)*](#)

47.2.2 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_DMABufferFlush\(\)*](#)
- [*HAL_UART_DMAResume\(\)*](#)
- [*HAL_UART_DMAStop\(\)*](#)
- [*HAL_UART_IRQHandler\(\)*](#)
- [*HAL_UART_TxCpltCallback\(\)*](#)
- [*HAL_UART_TxHalfCpltCallback\(\)*](#)

- [HAL_UART_RxCpltCallback\(\)](#)
- [HAL_UART_RxHalfCpltCallback\(\)](#)
- [HAL_UART_ErrorCallback\(\)](#)

47.2.3 Peripheral Control functions

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

- HAL_MultiProcessor_EnableMuteMode() API enables mute mode
- HAL_MultiProcessor_DisableMuteMode() API disables mute mode
- HAL_MultiProcessor_EnterMuteMode() API enters mute mode
- HAL_HalfDuplex_EnableTransmitter() API enables the transmitter
- HAL_HalfDuplex_EnableReceiver() API enables the receiver
- HAL_UART_GetState() API is helpful to check in run-time the state of the UART peripheral
- HAL_UART_GetError() API is helpful to check in run-time the error state of the UART peripheral

This section contains the following APIs:

- [HAL_MultiProcessor_EnableMuteMode\(\)](#)
- [HAL_MultiProcessor_DisableMuteMode\(\)](#)
- [HAL_MultiProcessor_EnterMuteMode\(\)](#)
- [HAL_HalfDuplex_EnableTransmitter\(\)](#)
- [HAL_HalfDuplex_EnableReceiver\(\)](#)
- [HAL_LIN_SendBreak\(\)](#)
- [HAL_UART_GetState\(\)](#)
- [HAL_UART_GetError\(\)](#)

47.2.4 HAL_UART_Init

Function Name	HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)
Function Description	Initializes the UART mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle .
Parameters	• huart: uart handle
Return values	• HAL: status

47.2.5 HAL_HalfDuplex_Init

Function Name	HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)
Function Description	Initializes the half-duplex mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle .
Parameters	• huart: uart handle
Return values	• HAL: status

47.2.6 HAL_LIN_Init

Function Name	HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)

Function Description	Initializes the LIN mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle .
Parameters	• 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	• HAL: status

47.2.7 HAL_MultiProcessor_Init

Function Name	HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)
Function Description	Initializes the multiprocessor mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.
Parameters	• 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	• HAL: status
Notes	• 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().

47.2.8 HAL_UART_DeInit

Function Name	HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)
Function Description	DeInitializes the UART peripheral.
Parameters	• huart: uart handle
Return values	• HAL: status

47.2.9 HAL_UART_MspInit

Function Name	void HAL_UART_MspInit (UART_HandleTypeDef * huart)
Function Description	UART MSP Init.
Parameters	• huart: uart handle

Return values

- **None:**

47.2.10 HAL_UART_MspDeInit

Function Name **void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)**

Function Description UART MSP DeInit.

Parameters

- **huart:** uart handle

Return values

- **None:**

47.2.11 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

- **huart:** uart handle
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

47.2.12 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

- **huart:** uart handle
- **pData:** pointer to data buffer
- **Size:** amount of data to be received
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

47.2.13 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

- **huart:** uart handle
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

47.2.14 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	• huart: uart handle • pData: pointer to data buffer • Size: amount of data to be received
Return values	• HAL: status

47.2.15 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	• huart: uart handle • pData: pointer to data buffer • Size: amount of data to be sent
Return values	• HAL: status

47.2.16 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	• huart: uart handle • pData: pointer to data buffer • Size: amount of data to be received
Return values	• HAL: status
Notes	• When the UART parity is enabled (PCE = 1) the data received contain the parity bit.

47.2.17 HAL_UART_DMAPause

Function Name	HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)
Function Description	Pauses the DMA Transfer.
Parameters	• huart: UART handle
Return values	• None:

47.2.18 HAL_UART_DMAResume

Function Name	HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)
Function Description	Resumes the DMA Transfer.
Parameters	• huart: UART handle
Return values	• None:

47.2.19 HAL_UART_DMAStop

Function Name	HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)
Function Description	Stops the DMA Transfer.
Parameters	• huart: UART handle
Return values	• None:

47.2.20	HAL_UART_IRQHandler
Function Name	void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)
Function Description	This function handles UART interrupt request.
Parameters	• huart: uart handle
Return values	• None:

47.2.21	HAL_UART_TxCpltCallback
Function Name	void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)
Function Description	Tx Transfer completed callbacks.
Parameters	• huart: uart handle
Return values	• None:

47.2.22	HAL_UART_TxHalfCpltCallback
Function Name	void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)
Function Description	Tx Half Transfer completed callbacks.
Parameters	• huart: UART handle
Return values	• None:

47.2.23	HAL_UART_RxCpltCallback
Function Name	void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)
Function Description	Rx Transfer completed callbacks.
Parameters	• huart: uart handle
Return values	• None:

47.2.24	HAL_UART_RxHalfCpltCallback
Function Name	void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)
Function Description	Rx Half Transfer completed callbacks.
Parameters	• huart: UART handle
Return values	• None:

47.2.25 HAL_UART_ErrorCallback

Function Name	void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)
Function Description	UART error callbacks.
Parameters	• huart: uart handle
Return values	• None:

47.2.26 HAL_MultiProcessor_EnableMuteMode

Function Name	HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)
Function Description	Enable UART in mute mode (doesn't mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called)
Parameters	• huart: uart handle
Return values	• HAL: status

47.2.27 HAL_MultiProcessor_DisableMuteMode

Function Name	HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)
Function Description	Disable UART mute mode (doesn't mean it actually wakes up the software, as it may not have been in mute mode at this very moment).
Parameters	• huart: uart handle
Return values	• HAL: status

47.2.28 HAL_MultiProcessor_EnterMuteMode

Function Name	void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)
Function Description	Enter UART mute mode (means UART actually enters mute mode).
Parameters	• huart: uart handle
Return values	• HAL: status

47.2.29 HAL_HalfDuplex_EnableTransmitter

Function Name	HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)
Function Description	Enables the UART transmitter and disables the UART receiver.
Parameters	• huart: UART handle
Return values	• HAL: status • None:

47.2.30 HAL_HalfDuplex_EnableReceiver

Function Name	HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)
Function Description	Enables the UART receiver and disables the UART transmitter.
Parameters	huart: UART handle
Return values	HAL: status

47.2.31 HAL_LIN_SendBreak

Function Name	HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)
Function Description	Transmits break characters.
Parameters	huart: pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	HAL: status

47.2.32 HAL_UART_GetState

Function Name	HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)
Function Description	return the UART state
Parameters	huart: uart handle
Return values	HAL: state

47.2.33 HAL_UART_GetError

Function Name	uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)
Function Description	Return the UART error code.
Parameters	huart: : pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	UART: Error Code

47.3 UART Firmware driver defines**47.3.1 UART*****UART advanced features initialization type definition***

```

UART_ADVFEATURE_NO_INIT
UART_ADVFEATURE_TXINVERT_INIT
UART_ADVFEATURE_RXINVERT_INIT
UART_ADVFEATURE_DATAINVERT_INIT
UART_ADVFEATURE_SWAP_INIT
UART_ADVFEATURE_RXOVERRUNDISABLE_INIT

```

UART_ADVFEATURE_DMADISABLEONERROR_INIT

UART_ADVFEATURE_AUTOBAUDRATE_INIT

UART_ADVFEATURE_MSBFIRST_INIT

IS_UART_ADVFEATURE_INIT

UART advanced auto baud rate activation definition

UART_ADVFEATURE_AUTOBAUDRATE_DISABLE

UART_ADVFEATURE_AUTOBAUDRATE_ENABLE

IS_UART_ADVFEATURE_AUTOBAUDRATE

UART CR1 DEAT address lsb position definition

UART_CR1_DEAT_ADDRESS_LSB_POS

UART CR1 DEDT address lsb position definition

UART_CR1_DEDT_ADDRESS_LSB_POS

UART CR2 address lsb position definition

UART_CR2_ADDRESS_LSB_POS

UART advanced data inv activation definition

UART_ADVFEATURE_DATAINV_DISABLE

UART_ADVFEATURE_DATAINV_ENABLE

IS_UART_ADVFEATURE_DATAINV

UART advanced DMA on Rx error activation definition

UART_ADVFEATURE_DMA_ENABLEONRXERROR

UART_ADVFEATURE_DMA_DISABLEONRXERROR

IS_UART_ADVFEATURE_DMAONRXERROR

UART DMA Rx definition

UART_DMA_RX_DISABLE

UART_DMA_RX_ENABLE

IS_UART_DMA_RX

UART DMA Tx definition

UART_DMA_TX_DISABLE

UART_DMA_TX_ENABLE

IS_UART_DMA_TX

UART driver polarity level definition

UART_DE_POLARITY_HIGH

UART_DE_POLARITY_LOW

IS_UART_DE_POLARITY

UART error definition

HAL_UART_ERROR_NONE No error

HAL_UART_ERROR_PE	Parity error
HAL_UART_ERROR_NE	Noise error
HAL_UART_ERROR_FE	frame error
HAL_UART_ERROR_ORE	Overrun error
HAL_UART_ERROR_DMA	DMA transfer error

UART Exported Macros**__HAL_UART_RESET_HANDLE_STATE****Description:**

- Reset UART handle state.

Parameters:

- __HANDLE__**: specifies the UART Handle. The Handle Instance which can be USART1, USART2 or LPUART.

Return value:

- None

__HAL_UART_FLUSH_DRREGISTER**Description:**

- Flush the UART Data registers.

Parameters:

- __HANDLE__**: specifies the UART Handle.

__HAL_UART_CLEAR_FLAG**Description:**

- Clears the specified UART pending flag.

Parameters:

- __HANDLE__**: specifies the UART Handle.
- __FLAG__**: specifies the flag to check. This parameter can be any combination of the following values:
 - UART_CLEAR_PEF
 - UART_CLEAR_FEF
 - UART_CLEAR_NEF
 - UART_CLEAR_OREF
 - UART_CLEAR_IDLEF
 - UART_CLEAR_TCF
 - UART_CLEAR_LBDF
 - UART_CLEAR_CTSF
 - UART_CLEAR_RTOF
 - UART_CLEAR_EOBF
 - UART_CLEAR_CMF
 - UART_CLEAR_WUF

Return value:

- None

__HAL_UART_CLEAR_PEF**Description:**

`__HAL_UART_CLEAR_FEFLAG`

- Clear the UART PE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

Description:

- Clear the UART FE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

Description:

- Clear the UART NE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

Description:

- Clear the UART ORE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

Description:

- Clear the UART IDLE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

Description:

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

Parameters:

`__HAL_UART_GET_FLAG`

- `__HANDLE__`: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `UART_FLAG_REACK`: Receive enable acknowledge flag
 - `UART_FLAG_TEACK`: Transmit enable acknowledge flag
 - `UART_FLAG_WUF`: Wake up from stop mode flag
 - `UART_FLAG_RWU`: Receiver wake up flag (is the UART in mute mode)
 - `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
 - `UART_FLAG_RTOF`: Receiver timeout flag
 - `UART_FLAG_CTS`: CTS Change flag (not available for UART4 and UART5)
 - `UART_FLAG_LBD`: LIN Break detection flag
 - `UART_FLAG_TXE`: Transmit data register empty flag
 - `UART_FLAG_TC`: Transmission Complete flag
 - `UART_FLAG_RXNE`: Receive data register not empty flag
 - `UART_FLAG_IDLE`: Idle Line detection flag
 - `UART_FLAG_ORE`: OverRun Error flag
 - `UART_FLAG_NE`: Noise Error flag
 - `UART_FLAG_FE`: Framing Error flag
 - `UART_FLAG_PE`: Parity Error flag

Return value:

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

Description:

- Enables the specified UART interrupt.

Parameters:`__HAL_UART_ENABLE_IT`

- `__HANDLE__`: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- `__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
 - `UART_IT_CM`: Character match interrupt
 - `UART_IT_CTS`: CTS change interrupt
 - `UART_IT_LBD`: LIN Break detection interrupt
 - `UART_IT_TXE`: Transmit Data Register empty interrupt
 - `UART_IT_TC`: Transmission complete interrupt
 - `UART_IT_RXNE`: Receive Data register not empty interrupt
 - `UART_IT_IDLE`: Idle line detection interrupt
 - `UART_IT_PE`: Parity Error interrupt
 - `UART_IT_ERR`: Error interrupt (Frame error, noise error, overrun error)

Return value:

- None

Description:

- Disables the specified UART interrupt.

Parameters:

- `__HANDLE__`: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- `__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
 - `UART_IT_CM`: Character match interrupt
 - `UART_IT_CTS`: CTS change interrupt
 - `UART_IT_LBD`: LIN Break detection interrupt
 - `UART_IT_TXE`: Transmit Data Register empty interrupt
 - `UART_IT_TC`: Transmission complete interrupt
 - `UART_IT_RXNE`: Receive Data

`__HAL_UART_DISABLE_IT`

- register not empty interrupt
- UART_IT_IDLE: Idle line detection interrupt
- UART_IT_PE: Parity Error interrupt
- UART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

Description:

- Checks whether the specified UART interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- `__IT__`: specifies the UART interrupt to check. This parameter can be one of the following values:
 - UART_IT_WUF: Wakeup from stop mode interrupt
 - UART_IT_CM: Character match interrupt
 - UART_IT_CTS: CTS change interrupt (not available for UART4 and UART5)
 - UART_IT_LBD: LIN Break detection interrupt
 - UART_IT_TXE: Transmit Data Register empty interrupt
 - UART_IT_TC: Transmission complete interrupt
 - UART_IT_RXNE: Receive Data register not empty interrupt
 - UART_IT_IDLE: Idle line detection interrupt
 - UART_IT_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).

Description:

- Checks whether the specified UART interrupt source is enabled.

`__HAL_UART_GET_IT``__HAL_UART_GET_IT_SOURCE`

Parameters:

- `__HANDLE__`: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- `__IT__`: specifies the UART interrupt source to check. This parameter can be one of the following values:
 - `UART_IT_CTS`: CTS change interrupt (not available for UART4 and UART5)
 - `UART_IT_LBD`: LIN Break detection interrupt
 - `UART_IT_TXE`: Transmit Data Register empty interrupt
 - `UART_IT_TC`: Transmission complete interrupt
 - `UART_IT_RXNE`: Receive Data register not empty interrupt
 - `UART_IT_IDLE`: Idle line detection interrupt
 - `UART_IT_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).

Description:

- Clears the specified UART ISR flag, in setting the proper ICR register flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- `__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

`__HAL_UART_CLEAR_IT`

- UART_CLEAR_TCF: Transmission Complete Clear Flag
- UART_CLEAR_LBDF: LIN Break Detection Clear Flag
- 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

Return value:

- None

Description:

- Set a specific UART request flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle. This parameter can be USART1, USART2 or LPUART.
- `__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

Return value:

- None

`__HAL_UART_SEND_REQ`**Description:**

- Enables the UART one bit sample method.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

`__HAL_UART_ONE_BIT_SAMPLE_ENABLE`**Description:**`__HAL_UART_ONE_BIT_SAMPLE_DISABLE`

<code>ABLE</code>	- Disables the UART one bit sample method. Parameters: <code>__HANDLE__</code>: specifies the UART Handle. Return value: - None
<code>__HAL_UART_ENABLE</code>	Description: - Enable UART. Parameters: <code>__HANDLE__</code>: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART. Return value: - None
<code>__HAL_UART_DISABLE</code>	Description: - Disable UART. Parameters: <code>__HANDLE__</code>: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART. Return value: - None
<code>__HAL_UART_HWCONTROL_CTS_ENABLE</code>	Description: - Enable CTS flow control This macro allows to enable CTS hardware flow control for a given UART instance, without need to call Parameters: <code>__HANDLE__</code>: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART. Return value: - None Notes: - 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_HWCONTROL_CTS_DISABLE

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__)).

Description:

- Disable CTS flow control This macro allows to disable CTS hardware flow control for a given UART instance, without need to call

Parameters:

- __HANDLE__: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

Return value:

- None

Notes:

- 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_RTS_ENABLE

Description:

- Enable RTS flow control This macro allows to enable RTS hardware flow control for a given UART instance, without need to call

Parameters:

- __HANDLE__: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

Return value:

- None

Notes:

- 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 This macro allows to disable RTS hardware flow control for a given USART instance, without need to call

Parameters:

- `__HANDLE__`: specifies the USART Handle. The Handle Instance can be USART1, USART2 or LPUART.

Return value:

- None

Notes:

- 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_ONE_BIT_ENABLE`

Description:

- macros to enable the USART's one bit sampling method

Parameters:

- `__HANDLE__`: specifies the USART Handle.

Return value:

- None

`__HAL_UART_ONE_BIT_DISABLE`

Description:

__DIV_LPUART

- macros to disable the UART's one bit sampling method

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Description:

- BRR division operation to set BRR register with LPUART.

Parameters:

- _PCLK_: LPUART clock
- _BAUD_: Baud rate set by the user

Return value:

- Division: result

Description:

- BRR division operation to set BRR register in 8-bit oversampling mode.

Parameters:

- _PCLK_: UART clock
- _BAUD_: Baud rate set by the user

Return value:

- Division: result

Description:

- BRR division operation to set BRR register in 16-bit oversampling mode.

Parameters:

- _PCLK_: UART clock
- _BAUD_: Baud rate set by the user

Return value:

- Division: result

Description:

- Check whether or not UART instance is Low Power UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- SET: (instance is LPUART) or RESET

UART_DIV_SAMPLING8

UART_DIV_SAMPLING16

UART_INSTANCE_LOWPPOWER

(instance isn't LPUART)

IS_UART_BAUDRATE

Description:

- Check UART Baud rate.

Parameters:

- BAUDRATE: Baudrate specified by the user. The maximum Baud Rate is derived from the maximum clock on L0 (i.e. 32 MHz) divided by the smallest oversampling used on the USART (i.e. 8).

Return value:

- Test: result (TRUE or FALSE).

IS_UART_7B_ADDRESS

Description:

- Check UART byte address.

Parameters:

- ADDRESS: UART 8-bit address for wake-up process scheme

Return value:

- Test: result (TRUE or FALSE).

IS_UART_4B_ADDRESS

Description:

- Check UART 4-bit address.

Parameters:

- ADDRESS: UART 4-bit address for wake-up process scheme

Return value:

- Test: result (TRUE or FALSE).

IS_UART_ASSERTIONTIME

Description:

- Check UART assertion time.

Parameters:

- TIME: 5-bit value assertion time

Return value:

- Test: result (TRUE or FALSE).

IS_UART_DEASSERTIONTIME

Description:

- Check UART deassertion time.

Parameters:

- TIME: 5-bit value deassertion time

Return value:

- Test: result (TRUE or FALSE).

UART flags definition

UART_FLAG_REACK	Receive Enable Acknowledge Flag
UART_FLAG_TEACK	Transmit Enable Acknowledge Flag
UART_FLAG_WUF	Wake Up from stop mode Flag
UART_FLAG_RWU	Receive Wake Up from mute mode Flag
UART_FLAG_SBKF	Send Break Flag
UART_FLAG_CMF	Character Match Flag
UART_FLAG_BUSY	Busy Flag
UART_FLAG_ABRF	Auto-Baud Rate Flag
UART_FLAG_ABRE	Auto-Baud Rate Error
UART_FLAG_EOBF	End Of Block Flag
UART_FLAG_RTOF	Receiver Time Out
UART_FLAG_CTS	CTS flag
UART_FLAG_CTSIF	CTS interrupt flag
UART_FLAG_LBDF	LIN Break Detection Flag
UART_FLAG_TXE	Transmit Data Register Empty
UART_FLAG_TC	Transmission Complete
UART_FLAG_RXNE	Read Data Register Not Empty
UART_FLAG_IDLE	IDLE line detected
UART_FLAG_ORE	OverRun Error
UART_FLAG_NE	Noise detected Flag
UART_FLAG_FE	Framing Error
UART_FLAG_PE	Parity Error

UART half duplex selection definition

UART_HALF_DUPLEX_DISABLE
UART_HALF_DUPLEX_ENABLE
IS_UART_HALF_DUPLEX

UART hardware flow control definition

UART_HWCONTROL_NONE
UART_HWCONTROL_RTS
UART_HWCONTROL_CTS
UART_HWCONTROL_RTS_CTS
IS_UART_HARDWARE_FLOW_CONTROL

UART interruption mask definition

UART_IT_MASK

UART interrupt 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_ERR
UART_IT_ORE
UART_IT_NE
UART_IT_FE

UART interrupt clear flags definition

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
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 LIN enable and disable definition

UART_LIN_DISABLE
UART_LIN_ENABLE
IS_UART_LIN

UART LIN break detection definition

UART_LINBREAKDETECTLENGTH_10B
UART_LINBREAKDETECTLENGTH_11B
IS_UART_LIN_BREAK_DETECT_LENGTH

UART mode definition

UART_MODE_RX
UART_MODE_TX

UART_MODE_TX_RX

IS_UART_MODE

UART advanced MSB first activation definition

UART_ADVFEATURE_MSBFIRST_DISABLE

UART_ADVFEATURE_MSBFIRST_ENABLE

IS_UART_ADVFEATURE_MSBFIRST

UART advanced mute mode activation definition

UART_ADVFEATURE_MUTEMODE_DISABLE

UART_ADVFEATURE_MUTEMODE_ENABLE

IS_UART_MUTE_MODE

UART one bit definition

UART_ONE_BIT_SAMPLE_DISABLE

UART_ONE_BIT_SAMPLE_ENABLE

IS_UART_ONE_BIT_SAMPLE

UART advanced overrun activation definition

UART_ADVFEATURE_OVERRUN_ENABLE

UART_ADVFEATURE_OVERRUN_DISABLE

IS_UART_OVERRUN

UART over sampling definition

UART_OVERSAMPLING_16

UART_OVERSAMPLING_8

IS_UART_OVERSAMPLING

UART parity definition

UART_PARITY_NONE

UART_PARITY_EVEN

UART_PARITY_ODD

IS_UART_PARITY

UART receiver timeOut definition

UART_RECEIVER_TIMEOUT_DISABLE

UART_RECEIVER_TIMEOUT_ENABLE

IS_UART_RECEIVER_TIMEOUT

UART request parameter definition

UART_AUTOBAUD_REQUEST Auto-Baud Rate Request

UART_SENDBREAK_REQUEST Send Break Request

UART_MUTE_MODE_REQUEST Mute Mode Request

UART_RXDATA_FLUSH_REQUEST Receive Data flush Request

UART_TXDATA_FLUSH_REQUEST Transmit data flush Request

IS_UART_REQUEST_PARAMETER

UART advanced Rx inv activation definition

UART_ADVFEATURE_RXINV_DISABLE

UART_ADVFEATURE_RXINV_ENABLE

IS_UART_ADVFEATURE_RXINV

UART advanced swap activation definition

UART_ADVFEATURE_SWAP_DISABLE

UART_ADVFEATURE_SWAP_ENABLE

IS_UART_ADVFEATURE_SWAP

UART state enable and disable definition

UART_STATE_DISABLE

UART_STATE_ENABLE

IS_UART_STATE

UART stop bit definition

UART_STOPBITS_1 USART frame with 1 stop bit

UART_STOPBITS_1_5 USART frame with 1.5 stop bits

UART_STOPBITS_2 USART frame with 2 stop bits

IS_UART_STOPBITS

IS_LPUART_STOPBITS

UART advanced stop mode activation definition

UART_ADVFEATURE_STOPMODE_DISABLE

UART_ADVFEATURE_STOPMODE_ENABLE

IS_UART_ADVFEATURE_STOPMODE

UART advanced Tx inv activation definition

UART_ADVFEATURE_TXINV_DISABLE

UART_ADVFEATURE_TXINV_ENABLE

IS_UART_ADVFEATURE_TXINV

UART wake up mode selection definition

UART_WAKEUP_ON_ADDRESS

UART_WAKEUP_ON_STARTBIT

UART_WAKEUP_ON_READDATA_NONEMPTY

IS_UART_WAKEUP_SELECTION

48 HAL UART Extension Driver

48.1 UARTEEx Firmware driver registers structures

48.1.1 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 [UARTEEx_WakeUp_Address_Length](#)
- *uint8_t UART_WakeUpTypeDef::Address*
UART/USART node address (7-bit long max)

48.2 UARTEEx Firmware driver API description

48.2.1 Initialization and Configuration functions

The `HAL_RS485Ex_Init()` API follows respectively the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [HAL_RS485Ex_Init\(\)](#)

48.2.2 Peripheral Control functions

This section provides functions allowing to:

- `UART_AdvFeatureConfig()` API optionally configures the UART advanced features
- `HAL_MultiProcessorEx_AddressLength_Set()` API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- `HAL_UARTEEx_EnableStopMode()` API enables the UART to wake up the MCU from stop mode
- `HAL_UARTEEx_DisableStopMode()` API disables the above functionality
- `HAL_UARTEEx_EnableClockStopMode()` API enables the UART HSI clock during stop mode
- `HAL_UARTEEx_DisableClockStopMode()` API disables the above functionality
- `UART_Wakeup_AddressConfig()` API configures the wake-up from stop mode parameters

This section contains the following APIs:

- [HAL_UARTEx_EnableStopMode\(\)](#)
- [HAL_UARTEx_EnableClockStopMode\(\)](#)
- [HAL_UARTEx_DisableStopMode\(\)](#)
- [HAL_UARTEx_DisableClockStopMode\(\)](#)
- [HAL_UARTEx_StopModeWakeUpSourceConfig\(\)](#)
- [HAL_MultiProcessorEx_AddressLength_Set\(\)](#)
- [HAL_UARTEx_WakeupCallback\(\)](#)

48.2.3 HAL_RS485Ex_Init

Function Name	HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)
Function Description	Initializes the RS485 Driver enable feature according to the specified parameters in the UART_InitTypeDef and creates the associated handle .
Parameters	• 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	• HAL: status

48.2.4 HAL_UARTEx_EnableStopMode

Function Name	HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (UART_HandleTypeDef * huart)
Function Description	Enable UART Stop Mode The UART is able to wake up the MCU from Stop mode as long as UART clock is HSI or LSE.
Parameters	• huart: uart handle
Return values	• HAL: status

48.2.5 HAL_UARTEx_EnableClockStopMode

Function Name	HAL_StatusTypeDef HAL_UARTEx_EnableClockStopMode (UART_HandleTypeDef * huart)
Function Description	Enable UART Clock in Stop Mode The UART keeps the Clock ON during Stop mode.
Parameters	• huart: uart handle

Return values

- **HAL:** status

48.2.6 HAL_UARTEx_DisableStopMode

Function Name **HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (UART_HandleTypeDef * huart)**

Function Description Disable UART Stop Mode.

Parameters

- **huart:** uart handle

Return values

- **HAL:** status

48.2.7 HAL_UARTEx_DisableClockStopMode

Function Name **HAL_StatusTypeDef HAL_UARTEx_DisableClockStopMode (UART_HandleTypeDef * huart)**

Function Description Disable UART Clock in Stop Mode.

Parameters

- **huart:** uart handle

Return values

- **HAL:** status

48.2.8 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

- **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_ADDRESS, UART_WAKEUP_ON_STARTBIT, UART_WAKEUP_ON_READDATA_NONEMPTY

Return values

- **HAL:** status

48.2.9 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.

48.2.10 HAL_UARTEx_WakeupCallback

Function Name **void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)**

Function Description UART wakeup from Stop mode callback.

Parameters	• huart: uart handle
Return values	• None:

48.3 UARTEEx Firmware driver defines

48.3.1 UARTEEx

Auto baud rate mode definition

UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT
 UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE
 UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFRAME
 UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME
 IS_UART_ADVFEATURE_AUTOBAUDRATEMODE

UARTEEx Exported Macros

UART_GETCLOCKSOURCE	Description: • Reports the UART clock source. Parameters: • <code>__HANDLE__</code>: specifies the UART Handle • <code>__CLOCKSOURCE__</code>: : output variable Return value: • UART: clocking source, written in <code>__CLOCKSOURCE__</code>.
UART_MASK_COMPUTATION	Description: • Reports the UART mask to apply to retrieve the received data according to the word length and to the parity bits activation. Parameters: • <code>__HANDLE__</code>: specifies the UART Handle Return value: • mask: to apply to UART RDR register value.

WakeUp address length definition

UART_ADDRESS_DETECT_4B
 UART_ADDRESS_DETECT_7B
 IS_UART_ADDRESSELENGTH_DETECT

Wakeup methods definition

UART_WAKEUPMETHOD_IDLELINE
 UART_WAKEUPMETHOD_ADDRESSMARK
 IS_UART_WAKEUPMETHOD

Word length definition

UART_WORDLENGTH_7B

UART_WORDLENGTH_8B

UART_WORDLENGTH_9B

IS_UART_WORD_LENGTH

49 HAL USART Generic Driver

49.1 USART Firmware driver registers structures

49.1.1 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](#)

49.1.2 USART_HandleTypeDef

Data Fields

- *USART_TypeDef * Instance*
- *USART_InitTypeDef Init*
- *uint8_t * pTxBuffPtr*
- *uint16_t TxXferSize*
- *__IO uint16_t TxXferCount*
- *uint8_t * pRxBuffPtr*
- *uint16_t RxXferSize*
- *__IO uint16_t RxXferCount*
- *uint16_t Mask*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *__IO HAL_USART_StateTypeDef State*
- *__IO uint32_t ErrorCode*

Field Documentation

- ***USART_TypeDef* USART_HandleTypeDef::Instance***
USART registers base address
- ***USART_InitTypeDef USART_HandleTypeDef::Init***
Usart communication parameters
- ***uint8_t* USART_HandleTypeDef::pTxBuffPtr***
Pointer to Usart Tx transfer Buffer
- ***uint16_t USART_HandleTypeDef::TxXferSize***
Usart Tx Transfer size
- ***__IO uint16_t USART_HandleTypeDef::TxXferCount***
Usart Tx Transfer Counter
- ***uint8_t* USART_HandleTypeDef::pRxBuffPtr***
Pointer to Usart Rx transfer Buffer
- ***uint16_t USART_HandleTypeDef::RxXferSize***
Usart Rx Transfer size
- ***__IO uint16_t USART_HandleTypeDef::RxXferCount***
Usart Rx Transfer Counter
- ***uint16_t USART_HandleTypeDef::Mask***
- ***DMA_HandleTypeDef* USART_HandleTypeDef::hdmatx***
Usart Tx DMA Handle parameters
- ***DMA_HandleTypeDef* USART_HandleTypeDef::hdmarx***
Usart Rx DMA Handle parameters
- ***HAL_LockTypeDef USART_HandleTypeDef::Lock***
Locking object
- ***__IO HAL_USART_StateTypeDef USART_HandleTypeDef::State***
Usart communication state
- ***__IO uint32_t USART_HandleTypeDef::ErrorCode***
USART Error code

49.2 USART Firmware driver API description

49.2.1 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Stop Bit
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), the possible USART frame formats are as listed in the following table.
 - USART polarity
 - USART phase
 - USART LastBit
 - Receiver/transmitter modes

Table 22: USART frame formats

M0 bit	PCE bit	USART frame
0	0	SB 8 bit data STB
0	1	SB 7 bit data PB STB
1	0	SB 9 bit data STB
1	1	SB 8 bit data PB STB

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual (RM0329)).

This section contains the following APIs:

- [*HAL_USART_Init\(\)*](#)
- [*HAL_USART_DeInit\(\)*](#)
- [*HAL_USART_MspInit\(\)*](#)
- [*HAL_USART_MspDeInit\(\)*](#)

49.2.2 IO operation functions

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

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA. These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive

process The HAL_USART_ErrorCallback() user callback will be executed when a communication error is detected.

2. Blocking mode API's are :
 - HAL_USART_Transmit() in simplex mode
 - HAL_USART_Receive() in full duplex receive only
 - HAL_USART_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
 - HAL_USART_Transmit_IT() in simplex mode
 - HAL_USART_Receive_IT() in full duplex receive only
 - HAL_USART_TransmitReceive_IT() in full duplex mode
 - HAL_USART_IRQHandler()
4. No-Blocking mode functions with DMA are :
 - HAL_USART_Transmit_DMA() in simplex mode
 - HAL_USART_Receive_DMA() in full duplex receive only
 - HAL_USART_TransmitReceive_DMA() in full duplex mode
 - HAL_USART_DMAPause()
 - HAL_USART_DMAResume()
 - HAL_USART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in No_Blocking mode:
 - HAL_USART_TxCpltCallback()
 - HAL_USART_RxCpltCallback()
 - HAL_USART_TxHalfCpltCallback()
 - HAL_USART_RxHalfCpltCallback()
 - HAL_USART_ErrorCallback()
 - HAL_USART_TxRxCpltCallback()

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\(\)*](#)

49.2.3 Peripheral State functions

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

- HAL_USART_GetState() API can be helpful to check in run-time the state of the USART peripheral.

- HAL_USART_GetError() API can be helpful to check in run-time the Error Code of the USART peripheral.
- USART_SetConfig() API is used to set the USART communication parameters.
- USART_CheckIdleState() API ensures that TEACK and/or REACK bits are set after initialization

This section contains the following APIs:

- [HAL_USART_GetState\(\)](#)
- [HAL_USART_GetError\(\)](#)

49.2.4 HAL_USART_Init

Function Name	HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)
Function Description	Initializes the USART mode according to the specified parameters in the USART_InitTypeDef and create the associated handle.
Parameters	• husart: USART handle
Return values	• HAL: status

49.2.5 HAL_USART_DeInit

Function Name	HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)
Function Description	DeInitializes the USART peripheral.
Parameters	• husart: USART handle
Return values	• HAL: status

49.2.6 HAL_USART_MspInit

Function Name	void HAL_USART_MspInit (USART_HandleTypeDef * husart)
Function Description	USART MSP Init.
Parameters	• husart: USART handle
Return values	• None:

49.2.7 HAL_USART_MspDeInit

Function Name	void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)
Function Description	USART MSP DeInit.
Parameters	• husart: USART handle
Return values	• None:

49.2.8 HAL_USART_Transmit

Function Name	HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)

Function Description Simplex Send an amount of data in blocking mode.

Parameters

- **husart:** USART handle
- **pTxData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

49.2.9 HAL_USART_Receive

Function Name **HAL_StatusTypeDef HAL_USART_Receive**
(USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t
Size, uint32_t Timeout)

Function Description Receive an amount of data in blocking mode To receive
synchronous data, dummy data are simultaneously transmitted.

Parameters

- **husart:** USART handle
- **pRxData:** pointer to data buffer
- **Size:** amount of data to be received
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

49.2.10 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

- **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

- **HAL:** status

49.2.11 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

- **husart:** USART handle
- **pTxData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

49.2.12 HAL_USART_Receive_IT

Function Name	HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	• husart: usart handle • pRxData: pointer to data buffer • Size: amount of data to be received
Return values	• HAL: status
49.2.13 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	• 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	• HAL: status
49.2.14 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	• husart: USART handle • pTxData: pointer to data buffer • Size: amount of data to be sent
Return values	• HAL: status
49.2.15 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	• husart: USART handle • pRxData: pointer to data buffer • Size: amount of data to be received
Return values	• HAL: status
Notes	• When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)

- The USART DMA transmit stream must be configured in order to generate the clock for the slave.

49.2.16 HAL_USART_TransmitReceive_DMA

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Full-Duplex Transmit Receive an amount of data in non blocking mode.
Parameters	• husart: 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.

49.2.17 HAL_USART_DMAPause

Function Name	HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)
Function Description	Pauses the DMA Transfer.
Parameters	• husart: USART handle
Return values	• None:

49.2.18 HAL_USART_DMAResume

Function Name	HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)
Function Description	Resumes the DMA Transfer.
Parameters	• husart: USART handle
Return values	• None:

49.2.19 HAL_USART_DMAStop

Function Name	HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)
Function Description	Stops the DMA Transfer.
Parameters	• husart: USART handle
Return values	• None:

49.2.20 HAL_USART_IRQHandler

Function Name	void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)

Function Description	This function handles USART interrupt request.
Parameters	• husart: USART handle
Return values	• None:

49.2.21 HAL_USART_TxCpltCallback

Function Name	void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx Transfer completed callbacks.
Parameters	• husart: USART handle
Return values	• None:

49.2.22 HAL_USART_TxHalfCpltCallback

Function Name	void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx Half Transfer completed callbacks.
Parameters	• husart: USART handle
Return values	• None:

49.2.23 HAL_USART_RxCpltCallback

Function Name	void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)
Function Description	Rx Transfer completed callbacks.
Parameters	• husart: USART handle
Return values	• None:

49.2.24 HAL_USART_RxHalfCpltCallback

Function Name	void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)
Function Description	Rx Half Transfer completed callbacks.
Parameters	• husart: USART handle
Return values	• None:

49.2.25 HAL_USART_TxRxCpltCallback

Function Name	void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx/Rx Transfers completed callback for the non-blocking process.
Parameters	• husart: USART handle
Return values	• None:

49.2.26 HAL_USART_ErrorCallback

Function Name	void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)
Function Description	USART error callbacks.
Parameters	• husart: USART handle
Return values	• None:

49.2.27 HAL_USART_GetState

Function Name	HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)
Function Description	Returns the USART state.
Parameters	• husart: USART handle
Return values	• HAL: state

49.2.28 HAL_USART_GetError

Function Name	uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)
Function Description	Return the USART error code.
Parameters	• husart: : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.
Return values	• USART: Error Code

49.3 USART Firmware driver defines

49.3.1 USART

USART clock activation definition

USART_CLOCK_DISABLE

USART_CLOCK_ENABLE

IS_USART_CLOCK

USART clock phase definition

USART_PHASE_1EDGE

USART_PHASE_2EDGE

IS_USART_PHASE

USART polarity level definition

USART_POLARITY_LOW

USART_POLARITY_HIGH

IS_USART_POLARITY

USART error definition

HAL_USART_ERROR_NONE No error

HAL_USART_ERROR_PE	Parity error
HAL_USART_ERROR_NE	Noise error
HAL_USART_ERROR_FE	frame error
HAL_USART_ERROR_ORE	Overrun error
HAL_USART_ERROR_DMA	DMA transfer error

USART Exported Macros**__HAL_USART_RESET_HANDLE_STATE****Description:**

- Reset USART handle state.

Parameters:

- __HANDLE__**: specifies the USART Handle. The Handle Instance which can be USART1 or USART2.

Return value:

- None

__HAL_USART_FLUSH_DRREGISTER**Description:**

- Flush the USART Data registers.

Parameters:

- __HANDLE__**: specifies the USART Handle.

__HAL_USART_GET_FLAG**Description:**

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

Parameters:

- __HANDLE__**: specifies the USART Handle which can be USART1 or USART2.
- __FLAG__**: specifies the flag to check. This parameter can be one of the following values:
 - USART_FLAG_REACK: Receive enable acknowledge flag
 - USART_FLAG_TEACK: Transmit enable acknowledge flag
 - USART_FLAG_BUSY: Busy flag
 - USART_FLAG_CTS: CTS Change flag
 - USART_FLAG_TXE: Transmit data register empty flag
 - USART_FLAG_TC: Transmission Complete flag
 - USART_FLAG_RXNE: Receive data register not empty flag
 - USART_FLAG_IDLE: Idle Line detection flag
 - USART_FLAG_ORE: OverRun Error

- flag
- USART_FLAG_NE: Noise Error flag
- USART_FLAG_FE: Framing Error flag
- USART_FLAG_PE: Parity Error flag

Return value:

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

Description:

- Clears 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_LBDF
 - USART_CLEAR_CTSF
 - USART_CLEAR_RTOF
 - USART_CLEAR_EOBF
 - USART_CLEAR_CMF
 - USART_CLEAR_WUF

Return value:

- None

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

__HAL_USART_CLEAR_FLAG

__HAL_USART_CLEAR_PEF

__HAL_USART_CLEAR_FEFLAG

__HAL_USART_CLEAR_NEFLAG

- None

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:**

- Enables the specified USART interrupt.

Parameters:

- **__HANDLE__**: specifies the USART Handle which can be USART1 or USART2.
- **__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)

`__HAL_USART_DISABLE_IT`

Return value:

- None

Description:

- Disables the specified USART interrupt.

Parameters:

- `__HANDLE__`: specifies the USART Handle which can be USART1 or USART2.
- `__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`

Description:

- Checks whether the specified USART interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the USART Handle which can be USART1 or USART2.
- `__IT__`: specifies the USART interrupt source to check. This parameter can be one of the following values:
 - `USART_IT_TXE`: Transmit Data Register empty interrupt
 - `USART_IT_TC`: Transmission complete interrupt
 - `USART_IT_RXNE`: Receive Data register not empty interrupt
 - `USART_IT_IDLE`: Idle line detection interrupt
 - `USART_IT_ORE`: OverRun Error interrupt
 - `USART_IT_NE`: Noise Error interrupt

- USART_IT_FE: Framing Error interrupt
- USART_IT_PE: Parity Error interrupt

Return value:

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

__HAL_USART_GET_IT_SOURCE**Description:**

- Checks whether the specified USART interrupt source is enabled.

Parameters:

- __HANDLE__: specifies the USART Handle which can be USART1 or USART2.
- __IT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
 - USART_IT_TXE: Transmit Data Register empty interrupt
 - USART_IT_TC: Transmission complete interrupt
 - USART_IT_RXNE: Receive Data register not empty interrupt
 - USART_IT_IDLE: Idle line detection interrupt
 - USART_IT_ORE: OverRun Error interrupt
 - USART_IT_NE: Noise Error interrupt
 - USART_IT_FE: Framing Error interrupt
 - USART_IT_PE: Parity Error interrupt

Return value:

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

__HAL_USART_CLEAR_IT**Description:**

- Clears the specified USART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the USART Handle which can be USART1 or USART2.
- __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 which can be USART1 or USART2.
- **__REQ__**: specifies the request flag to set This parameter can be one of the following values:
 - USART_RXDATA_FLUSH_REQUEST: Receive Data flush Request
 - USART_TXDATA_FLUSH_REQUEST: Transmit data flush Request

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_ENABLE**Description:**

- Enables the USART one bit sample method.

Parameters:

- **__HANDLE__**: specifies the USART Handle.

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_DISABLE**Description:**

- Disables the UART one bit sample method.

Parameters:

- **__HANDLE__**: specifies the UART Handle.

Return value:

- None

__HAL_USART_ENABLE**Description:**

- Enable USART.

Parameters:

- **__HANDLE__**: specifies the USART Handle. The Handle Instance which can be USART1 or USART2.

Return value:

- None

__HAL_USART_DISABLE**Description:**

- Disable USART.

Parameters:

- **__HANDLE__**: specifies the USART Handle. The Handle Instance which can be USART1 or USART2.

Return value:

- None

IS_USART_BAUDRATE**Description:**

- Check USART Baud rate.

Parameters:

- **BAUDRATE**: Baudrate specified by the user The maximum Baud Rate is derived from the maximum clock on L0 (i.e. 32 MHz) divided by the smallest oversampling used on the USART (i.e. 8)

Return value:

- Test: result (TRUE or FALSE).

USART flag definitions

USART_FLAG_REACK	Receive Enable Acknowledge Flag
USART_FLAG_TEACK	Transmit Enable Acknowledge Flag
USART_FLAG_BUSY	Busy Flag
USART_FLAG_CTS	CTS flag
USART_FLAG_CTSIF	CTS interrupt flag
USART_FLAG_LBDF	LIN Break Detection Flag
USART_FLAG_TXE	Transmit Data Register Empty
USART_FLAG_TC	Transmission Complete
USART_FLAG_RXNE	Read Data Register Not Empty
USART_FLAG_IDLE	IDLE line detected
USART_FLAG_ORE	OverRun Error
USART_FLAG_NE	Noise detected Flag

USART_FLAG_FE Framing Error

USART_FLAG_PE Parity Error

USART interruption mask definition

USART_IT_MASK

USART interrupt definition

USART_IT_PE

USART_IT_TXE

USART_IT_TC

USART_IT_RXNE

USART_IT_IDLE

USART_IT_ERR

USART_IT_ORE

USART_IT_NE

USART_IT_FE

USART interrupt clear flags definition

USART_CLEAR_PEF Parity Error Clear Flag

USART_CLEAR_FEF Framing Error Clear Flag

USART_CLEAR_NEF Noise detected Clear Flag

USART_CLEAR_OREF OverRun Error Clear Flag

USART_CLEAR_IDLEF IDLE line detected Clear Flag

USART_CLEAR_TCF Transmission Complete Clear Flag

USART_CLEAR_CTSF CTS Interrupt Clear Flag

USART last bit activation definition

USART_LASTBIT_DISABLE

USART_LASTBIT_ENABLE

IS_USART_LASTBIT

USART mode definition

USART_MODE_RX

USART_MODE_TX

USART_MODE_TX_RX

IS_USART_MODE

USART parity definition

USART_PARITY_NONE

USART_PARITY_EVEN

USART_PARITY_ODD

IS_USART_PARITY

USART request parameter definition

USART_RXDATA_FLUSH_REQUEST	Receive Data flush Request
USART_TXDATA_FLUSH_REQUEST	Transmit data flush Request
IS_USART_REQUEST_PARAMETER	

USART stop bit definition

USART_STOPBITS_1
USART_STOPBITS_2
USART_STOPBITS_1_5
IS_USART_STOPBITS

50 HAL USART Extension Driver

50.1 USARTEEx Firmware driver defines

50.1.1 USARTEEx

USARTEEx Exported Macros

USART_GETCLOCKSOURCE

Description:

- Reports the USART clock source.

Parameters:

- `__HANDLE__`: specifies the USART Handle
- `__CLOCKSOURCE__`: output variable

Return value:

- the: USART clocking source, written in `__CLOCKSOURCE__`.

USART_MASK_COMPUTATION

Description:

- Reports the USART mask to apply to retrieve the received data according to the word length and to the parity bits activation.

Parameters:

- `__HANDLE__`: specifies the USART Handle

Return value:

- mask: to apply to USART RDR register value.

Word length definition

USART_WORDLENGTH_7B

USART_WORDLENGTH_8B

USART_WORDLENGTH_9B

IS_USART_WORD_LENGTH

51 HAL WWDG Generic Driver

51.1 WWDG Firmware driver registers structures

51.1.1 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

51.1.2 WWDG_HandleTypeDef

Data Fields

- *WWDG_TypeDef * Instance*
- *WWDG_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_WWDG_StateTypeDef State*

Field Documentation

- *WWDG_TypeDef* WWDG_HandleTypeDef::Instance*
Register base address
- *WWDG_InitTypeDef WWDG_HandleTypeDef::Init*
WWDG required parameters
- *HAL_LockTypeDef WWDG_HandleTypeDef::Lock*
WWDG locking object
- *__IO HAL_WWDG_StateTypeDef WWDG_HandleTypeDef::State*
WWDG communication state

51.2 WWDG Firmware driver API description

51.2.1 WWDG specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls over from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.
- Once enabled the WWDG cannot be disabled except by a system reset.
- WWDGRST flag in RCC_CSR register can be used to inform when a WWDG reset occurs.
- The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.
- $\text{WWDG counter clock} = \text{PCLK1} / \text{Prescaler}$
 $\text{WWDG timeout} = (\text{WWDG counter clock}) * (\text{counter value})$
- Min-max timeout value @32 MHz(PCLK1): ~128.0 us / ~65.54 ms

51.2.2 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()`. Once enabled, EWI interrupt cannot be disabled except by a system reset.
- Then the application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset, using `HAL_WWDG_Refresh()` function. This operation must occur only when the counter is lower than the refresh window value already programmed.

WWDG HAL driver macros list

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

- `__HAL_WWDG_ENABLE`: Enable the WWDG peripheral
- `__HAL_WWDG_GET_FLAG`: Get the selected WWDG's flag status
- `__HAL_WWDG_CLEAR_FLAG`: Clear the WWDG's pending flags
- `__HAL_WWDG_ENABLE_IT`: Enables the WWDG early wakeup interrupt

51.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the WWDG according to the specified parameters in the `WWDG_InitTypeDef` and create the associated handle
- DeInitialize the WWDG peripheral
- Initialize the WWDG MSP
- DeInitialize the WWDG MSP

This section contains the following APIs:

- [HAL_WWDG_Init\(\)](#)
- [HAL_WWDG_DeInit\(\)](#)
- [HAL_WWDG_MspInit\(\)](#)
- [HAL_WWDG_MspDeInit\(\)](#)
- [HAL_WWDG_WakeupCallback\(\)](#)

51.2.4 IO operation functions

This section provides functions allowing to:

- Start the WWDG.
- Refresh the WWDG.
- Handle WWDG interrupt request.

This section contains the following APIs:

- [HAL_WWDG_Start\(\)](#)
- [HAL_WWDG_Start_IT\(\)](#)
- [HAL_WWDG_Refresh\(\)](#)
- [HAL_WWDG_IRQHandler\(\)](#)
- [HAL_WWDG_WakeupCallback\(\)](#)

51.2.5 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\(\)](#)

51.2.6 HAL_WWDG_Init

Function Name	HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwwdg)
Function Description	Initializes the WWDG according to the specified parameters in the WWDG_InitTypeDef and creates the associated handle.
Parameters	• hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• HAL: status

51.2.7 HAL_WWDG_DeInit

Function Name	HAL_StatusTypeDef HAL_WWDG_DeInit (WWDG_HandleTypeDef * hwwdg)
Function Description	DeInitializes the WWDG peripheral.
Parameters	• hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• HAL: status

51.2.8 HAL_WWDG_MspInit

Function Name	void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)

Function Description	Initializes the WWDG MSP.
Parameters	hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	None:

51.2.9 HAL_WWDG_MspDeInit

Function Name	void HAL_WWDG_MspDeInit (WWDG_HandleTypeDef * hwwdg)
Function Description	DeInitializes the WWDG MSP.
Parameters	hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	None:

51.2.10 HAL_WWDG_WakeupCallback

Function Name	void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwwdg)
Function Description	Early Wakeup WWDG callback.
Parameters	hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	None:

51.2.11 HAL_WWDG_Start

Function Name	HAL_StatusTypeDef HAL_WWDG_Start (WWDG_HandleTypeDef * hwwdg)
Function Description	Starts the WWDG.
Parameters	hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	HAL: status

51.2.12 HAL_WWDG_Start_IT

Function Name	HAL_StatusTypeDef HAL_WWDG_Start_IT (WWDG_HandleTypeDef * hwwdg)
Function Description	Starts the WWDG with interrupt enabled.
Parameters	hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	HAL: status

51.2.13 HAL_WWDG_Refresh



Function Name	HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg, uint32_t Counter)
Function Description	Refreshes the WWDG.
Parameters	• hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module. • Counter: value of counter to put in WWDG counter
Return values	• HAL: status

51.2.14 HAL_WWDG_IRQHandler

Function Name	void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)
Function Description	Handles WWDG interrupt request.
Parameters	• hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• None:
Notes	• The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled using <code>__HAL_WWDG_ENABLE_IT()</code> macro. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

51.2.15 HAL_WWDG_WakeupCallback

Function Name	void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwwdg)
Function Description	Early Wakeup WWDG callback.
Parameters	• hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• None:

51.2.16 HAL_WWDG_GetState

Function Name	HAL_WWDG_StateTypeDef HAL_WWDG_GetState (WWDG_HandleTypeDef * hwwdg)
Function Description	Returns the WWDG state.
Parameters	• hwwdg: : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• HAL: state

51.3 WWDG Firmware driver defines

51.3.1 WWDG

WWDG BitAddress AliasRegion

WWDG_CFR_BASE

WWDG Exported Constants

IS_WWDG_PRESCALER

IS_WWDG_WINDOW

IS_WWDG_COUNTER

WWDG Exported Macros

__HAL_WWDG_RESET_HANDLE_STATE

Description:

- Reset WWDG handle state.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_ENABLE

Description:

- Enables the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_DISABLE

Description:

- Disables the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment. Once enable, WWDG Peripheral cannot be disabled except by a system reset.

__HAL_WWDG_ENABLE_IT

Description:

- Enables the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle

__HAL_WWDG_DISABLE_IT

- **__INTERRUPT__**: specifies the interrupt to enable. This parameter can be one of the following values:
 - **WWDG_IT_EWI**: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

Description:

- Disables the WWDG early wakeup interrupt.

Parameters:

- **__HANDLE__**: WWDG handle:
- **__INTERRUPT__**: specifies the interrupt to disable.
 - **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:**

- Gets the selected WWDG's it status.

Parameters:

- **__HANDLE__**: WWDG handle
- **__INTERRUPT__**: specifies the it to check. This parameter can be one of the following values:
 - **WWDG_FLAG_EWIF**: Early wakeup interrupt IT

Return value:

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

Description:

- Clear the WWDG's interrupt pending bits bits to clear the selected interrupt pending bits.

Parameters:**__HAL_WWDG_CLEAR_IT**

`__HAL_WWDG_GET_FLAG`

- `__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

Description:

- Gets the selected WWDG's flag status.

Parameters:

- `__HANDLE__`: WWDG handle
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

Return value:

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

`__HAL_WWDG_CLEAR_FLAG`**Description:**

- Clears the WWDG's pending flags.

Parameters:

- `__HANDLE__`: WWDG handle
- `__FLAG__`: specifies the flag to clear. This parameter can be one of the following values:
 - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

Return value:

- None

`__HAL_WWDG_GET_IT_SOURCE`**Description:**

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

Parameters:

- `__HANDLE__`: WWDG Handle.
- `__INTERRUPT__`: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
 - `WWDG_IT_EWI`: Early Wakeup Interrupt

Return value:

- state: of `__INTERRUPT__` (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI

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

52 FAQs

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
 - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
 - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not required in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, ...). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL_UART_Init() then HAL_UART_Transmit() or HAL_UART_Receive().

Which STM32L0 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32L0 devices. To ensure compatibility between all devices and portability with others series and lines, the API is split into the generic and the extension APIs . For more details, please refer to [Section 3.4: "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: stm32l0xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration, Ethernet parameters configuration...)

A template is provided in the HAL drivers folders (stm32l0xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32l0xx_hal.h file has to be included.

What is the difference between stm32l0xx_hal_ppp.c/h and stm32l0xx_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 (stm32l0xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32l0xx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

Is it possible to use the APIs available in stm32l0xx_ll_ppp.c?

These APIs cannot be used directly because they are internal and offer services to upper layer drivers. As an example stm32l0xx_ll_fsmc.c/h driver is used by stm32l0xx_hal_sram.c, stm32l0xx_hal_nor.c, stm32l0xx_hal_nand.c and stm32l0xx_hal_sdram.c drivers.

Initialization and I/O operation functions

How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system_stm32l0xx.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 stm32l0xx_hal_msp.c. A template is provided in the HAL driver folders (stm32l0xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute `__weak`)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use `HAL_Init()` function at the beginning of the user application?

It is mandatory to use `HAL_Init()` function to enable the system configuration (Prefetch, Data instruction cache,...), configure the `systick` and the NVIC priority grouping and the hardware low level initialization.

The `systick` configuration shall be adjusted by calling **`HAL_RCC_ClockConfig()`** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling `HAL_IncTick()` function in SysTick ISR and retrieve the value of this variable by calling `HAL_GetTick()` function.

The call `HAL_GetTick()` function is mandatory when using HAL drivers with Polling Process or when using `HAL_Delay()`.

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could `HAL_Delay()` function block my application under certain conditions?

Care must be taken when using `HAL_Delay()` since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if `HAL_Delay()` is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use `HAL_NVIC_SetPriority()` function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call `HAL_Init()` function to initialize the system (data cache, NVIC priority,...).
2. Initialize the system clock by calling `HAL_RCC_OscConfig()` followed by `HAL_RCC_ClockConfig()`.
3. Add `HAL_IncTick()` function under `SysTick_Handler()` ISR function to enable polling process when using `HAL_Delay()` function
4. Start initializing your peripheral by calling `HAL_PPP_Init()`.
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling `HAL_PPP_MspInit()` in `stm32l0xx_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 stm32l0xx_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 stm32l0xx_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.

53 Revision history

Table 23: Document revision history

Date	Revision	Changes
03-Jun-2014	1	Initial release.
30-Jul-2015	2	Section 3.4: "Devices supported by HAL drivers" added support for STM32L07xxx and STM32L08xxx and updated stm32l0xx_hal_dac_ex.c and stm32l0xx_hal_lcd.c support for STM32L05/6xx devices. Removed note related to STM32Cube V1.0 in Section 3.12.2.1: "HAL global initialization". Updated common macros in Section 3.9: "HAL common resources" Updated GPIO_Typedef type (AF0) in Section 21: "HAL GPIO Generic Driver". Added new HAL_NVIC_GetPriority(IRQn_Type IRQn function in Section 9: "HAL CORTEX Generic Driver".
20-Nov-2015	3	Added stm32l0xx_flash.ld in User-application files. Added STM32L011xx, STM32L021xx, STM32L031xx and STM32L041xx in Section 3.4: "Devices supported by HAL drivers". Updated example of peripheral structure in Peripheral handle structures. Changed HAL_ADCEX_CalibrationStart() into HAL_ADCEX_Calibration_Start() in HAL extension model overview. Replaced GPIO_SPEED_LOW by GPIO_SPEED_FREQ_LOW, GPIO_SPEED_MEDIUM by GPIO_SPEED_FREQ_MEDIUM, GPIO_SPEED_FAST by GPIO_SPEED_FREQ_HIGH and GPIO_SPEED_HIGH by GPIO_SPEED_FREQ_VERY_HIGH) speed in Table 12: "Description of GPIO_InitTypeDef structure". Updated example of system clock configuration code in System clock initialization. Add new API to control the MPU (see HAL_MPU_Enable and HAL_MPU_Disable). Renamed some HAL macros and driver define statements to ensure consistency all over STM32 families: Added new define statements to specify the GPIO Speed (GPIO_SPEED_LOW replaced by GPIO_SPEED_FREQ_LOW for example). Added new macro to check the Comparator inputs (IS_COMP1_LPTIMCONNECTION and IS_COMP2_LPTIMCONNECTION).

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