

Square Root Approximation

0.1.-

Generated by Doxygen 1.9.8

1 Topic Index	1
1.1 Topics	1
2 Data Structure Index	5
2.1 Data Structures	5
3 File Index	7
3.1 File List	7
4 Topic Documentation	9
4.1 STM32F4xx_StdPeriph_Driver	9
4.1.1 Detailed Description	9
4.1.2 ADC	9
4.1.2.1 Detailed Description	12
4.1.2.2 Macro Definition Documentation	12
4.1.2.3 Function Documentation	14
4.1.2.4 ADC_Private_Functions	33
4.1.2.5 ADC_Exported_Constants	61
4.1.3 GPIO	87
4.1.3.1 Detailed Description	88
4.1.3.2 Macro Definition Documentation	89
4.1.3.3 Enumeration Type Documentation	89
4.1.3.4 Function Documentation	92
4.1.3.5 GPIO_Private_Functions	99
4.1.3.6 GPIO_Exported_Constants	110
4.1.4 I2C	122
4.1.4.1 Detailed Description	124
4.1.4.2 Macro Definition Documentation	124
4.1.4.3 Function Documentation	124
4.1.4.4 I2C_Private_Functions	140
4.1.4.5 I2C_Exported_Constants	162
4.1.5 RCC	179
4.1.5.1 Detailed Description	182
4.1.5.2 Macro Definition Documentation	182
4.1.5.3 Function Documentation	184
4.1.5.4 RCC_Private_Functions	212
4.1.5.5 RCC_Exported_Constants	244
4.1.6 USART	265
4.1.6.1 Detailed Description	266
4.1.6.2 Macro Definition Documentation	266
4.1.6.3 Function Documentation	267
4.1.6.4 USART_Private_Functions	278
4.1.6.5 USART_Exported_Constants	297

4.2 CMSIS	307
4.2.1 Detailed Description	308
4.2.2 Stm32f4xx_system	308
4.2.2.1 Detailed Description	308
4.2.2.2 STM32F4xx_System_Private_Includes	308
4.2.2.3 STM32F4xx_System_Private_TypesDefinitions	308
4.2.2.4 STM32F4xx_System_Private_Defines	308
4.2.2.5 STM32F4xx_System_Private_Macros	308
4.2.2.6 STM32F4xx_System_Private_Variables	308
4.2.2.7 STM32F4xx_System_Private_FunctionPrototypes	309
4.2.2.8 STM32F4xx_System_Private_Functions	309
5 Data Structure Documentation	311
5.1 _ADC_InitTypeDef Struct Reference	311
5.1.1 Field Documentation	311
5.1.1.1 ClockPrescaler	311
5.1.1.2 ContinuousConvMode	311
5.1.1.3 DataAlign	311
5.1.1.4 DiscontinuousConvMode	311
5.1.1.5 DMAContinuousRequests	312
5.1.1.6 EOCSelection	312
5.1.1.7 ExternalTrigConv	312
5.1.1.8 ExternalTrigConvEdge	312
5.1.1.9 NbrOfConversion	312
5.1.1.10 NbrOfDiscConversion	312
5.1.1.11 Resolution	312
5.1.1.12 ScanConvMode	312
5.2 ADC_ChannelConfTypeDef Struct Reference	312
5.2.1 Field Documentation	313
5.2.1.1 Channel	313
5.2.1.2 Offset	313
5.2.1.3 Rank	313
5.2.1.4 SamplingTime	313
5.3 ADC_CommonInitTypeDef Struct Reference	313
5.3.1 Detailed Description	313
5.3.2 Field Documentation	313
5.3.2.1 ADC_DMAAccessMode	313
5.3.2.2 ADC_Mode	314
5.3.2.3 ADC_Prescaler	314
5.3.2.4 ADC_TwoSamplingDelay	314
5.4 ADC_HandleTypeDef Struct Reference	314
5.4.1 Field Documentation	314

5.4.1.1 ErrorCode	314
5.4.1.2 Init	314
5.4.1.3 Instance	314
5.4.1.4 Lock	314
5.4.1.5 NbrOfCurrentConversionRank	314
5.4.1.6 State	315
5.5 ADC_InitTypeDef Struct Reference	315
5.5.1 Detailed Description	315
5.5.2 Field Documentation	315
5.5.2.1 ADC_ContinuousConvMode	315
5.5.2.2 ADC_DataAlign	315
5.5.2.3 ADC_ExternalTrigConv	315
5.5.2.4 ADC_ExternalTrigConvEdge	315
5.5.2.5 ADC_NbrOfConversion	315
5.5.2.6 ADC_Resolution	316
5.5.2.7 ADC_ScanConvMode	316
5.6 analogin_s Struct Reference	316
5.6.1 Field Documentation	316
5.6.1.1 adc	316
5.6.1.2 channel	316
5.6.1.3 pin	316
5.7 GPIO_InitTypeDef Struct Reference	316
5.7.1 Detailed Description	317
5.7.2 Field Documentation	317
5.7.2.1 Alternate	317
5.7.2.2 GPIO_Mode	317
5.7.2.3 GPIO_OType	317
5.7.2.4 GPIO_Pin	317
5.7.2.5 GPIO_PuPd	317
5.7.2.6 GPIO_Speed	317
5.7.2.7 Mode	317
5.7.2.8 Pin	317
5.7.2.9 Pull	317
5.7.2.10 Speed	317
5.8 I2C_InitTypeDef Struct Reference	318
5.8.1 Detailed Description	318
5.8.2 Field Documentation	318
5.8.2.1 I2C_Ack	318
5.8.2.2 I2C_AcknowledgedAddress	318
5.8.2.3 I2C_ClockSpeed	318
5.8.2.4 I2C_DutyCycle	318
5.8.2.5 I2C_Mode	318

5.8.2.6 I2C_OwnAddress1	318
5.9 PinMap Struct Reference	319
5.9.1 Field Documentation	319
5.9.1.1 function	319
5.9.1.2 peripheral	319
5.9.1.3 pin	319
5.10 RCC_ClocksTypeDef Struct Reference	319
5.10.1 Field Documentation	319
5.10.1.1 HCLK_Frequency	319
5.10.1.2 PCLK1_Frequency	319
5.10.1.3 PCLK2_Frequency	319
5.10.1.4 SYSCLK_Frequency	320
5.11 USART_ClockInitTypeDef Struct Reference	320
5.11.1 Detailed Description	320
5.11.2 Field Documentation	320
5.11.2.1 USART_Clock	320
5.11.2.2 USART_CPHA	320
5.11.2.3 USART_CPOL	320
5.11.2.4 USART_LastBit	320
5.12 USART_InitTypeDef Struct Reference	320
5.12.1 Detailed Description	321
5.12.2 Field Documentation	321
5.12.2.1 USART_BaudRate	321
5.12.2.2 USART_HardwareFlowControl	321
5.12.2.3 USART_Mode	321
5.12.2.4 USART_Parity	321
5.12.2.5 USART_StopBits	321
5.12.2.6 USART_WordLength	321
6 File Documentation	323
6.1 drivers/adc.c File Reference	323
6.1.1 Function Documentation	323
6.1.1.1 _ADC_ConfigChannel()	323
6.1.1.2 _ADC_GetValue()	324
6.1.1.3 _ADC_Init()	324
6.1.1.4 _ADC_PollForConversion()	324
6.1.1.5 _adc_read()	324
6.1.1.6 _ADC_Start()	324
6.1.1.7 _GPIO_Init()	324
6.1.1.8 adc_init()	324
6.1.1.9 adc_read()	324
6.1.1.10 analogin_init()	324

6.1.1.11 pin_function()	324
6.1.1.12 pinmap_find_function()	325
6.1.1.13 pinmap_find_peripheral()	325
6.1.1.14 pinmap_function()	325
6.1.1.15 pinmap_peripheral()	325
6.1.1.16 pinmap_pinout()	325
6.1.2 Variable Documentation	325
6.1.2.1 AdcHandle	325
6.1.2.2 PinMap_ADC	325
6.2 drivers/adc.h File Reference	325
6.2.1 Detailed Description	328
6.2.2 Macro Definition Documentation	328
6.2.2.1 _ADC_GET_FLAG	328
6.2.2.2 _IS_BIT_CLR	328
6.2.2.3 _IS_BIT_SET	328
6.2.2.4 ADC1_BASE	328
6.2.2.5 ADC_CHANNEL_0	328
6.2.2.6 ADC_CHANNEL_1	328
6.2.2.7 ADC_CHANNEL_10	328
6.2.2.8 ADC_CHANNEL_11	328
6.2.2.9 ADC_CHANNEL_12	329
6.2.2.10 ADC_CHANNEL_13	329
6.2.2.11 ADC_CHANNEL_14	329
6.2.2.12 ADC_CHANNEL_15	329
6.2.2.13 ADC_CHANNEL_16	329
6.2.2.14 ADC_CHANNEL_17	329
6.2.2.15 ADC_CHANNEL_18	329
6.2.2.16 ADC_CHANNEL_2	329
6.2.2.17 ADC_CHANNEL_3	329
6.2.2.18 ADC_CHANNEL_4	329
6.2.2.19 ADC_CHANNEL_5	329
6.2.2.20 ADC_CHANNEL_6	329
6.2.2.21 ADC_CHANNEL_7	329
6.2.2.22 ADC_CHANNEL_8	329
6.2.2.23 ADC_CHANNEL_9	329
6.2.2.24 ADC_CHANNEL_TEMPSENSOR	330
6.2.2.25 ADC_CHANNEL_VBAT	330
6.2.2.26 ADC_CHANNEL_VREFINT	330
6.2.2.27 ADC_CR1_DISCONTINUOUS	330
6.2.2.28 ADC_CR1_SCANCONV	330
6.2.2.29 ADC_CR2_CONTINUOUS	330
6.2.2.30 ADC_CR2_DMAContReq	330

6.2.2.31	ADC_CR2_EOCSelection	330
6.2.2.32	ADC_SMPR1	330
6.2.2.33	ADC_SMPR2	330
6.2.2.34	ADC_SOFTWARE_START	330
6.2.2.35	ADC_SQR1	330
6.2.2.36	ADC_SQR1_RK	331
6.2.2.37	ADC_SQR2_RK	331
6.2.2.38	ADC_SQR3_RK	331
6.2.2.39	ADC_STAB_DELAY_US	331
6.2.2.40	GPIO_MODE	331
6.2.2.41	GPIO_NOPULL	331
6.2.2.42	GPIO_PULLDOWN	331
6.2.2.43	GPIO_PULLUP	331
6.2.2.44	GPIO_SPEED_FAST	331
6.2.2.45	GPIO_SPEED_HIGH	331
6.2.2.46	GPIO_SPEED_LOW	331
6.2.2.47	GPIO_SPEED_MEDIUM	332
6.2.2.48	GPIOA_BASE	332
6.2.2.49	RCC_ADC1_CLK_ENABLE	332
6.2.2.50	RCC_GPIOA_CLK_ENABLE	332
6.2.2.51	RCC_GPIOB_CLK_ENABLE	332
6.2.2.52	RCC_GPIOC_CLK_ENABLE	332
6.2.2.53	STM_MODE_ANALOG	332
6.2.2.54	STM_PIN	332
6.2.2.55	STM_PIN_AFNUM	333
6.2.2.56	STM_PIN_CHANNEL	333
6.2.2.57	STM_PIN_DATA_EXT	333
6.2.2.58	STM_PIN_MODE	333
6.2.2.59	STM_PIN_PUPD	333
6.2.2.60	STM_PORT	333
6.2.2.61	UNUSED	333
6.2.3	Enumeration Type Documentation	333
6.2.3.1	ADCName	333
6.2.3.2	HAL_ADC_StateTypeDef	333
6.2.3.3	HAL_LockTypeDef	334
6.2.3.4	PortName	334
6.2.4	Function Documentation	334
6.2.4.1	_ADC_ConfigChannel()	334
6.2.4.2	_ADC_GetValue()	334
6.2.4.3	_ADC_Init()	334
6.2.4.4	_ADC_PollForConversion()	335
6.2.4.5	_adc_read()	335

6.2.4.6 _ADC_Start()	335
6.2.4.7 _GPIO_Init()	335
6.2.4.8 adc_init()	335
6.2.4.9 adc_read()	335
6.2.4.10 analogin_init()	335
6.2.4.11 pin_function()	335
6.2.4.12 pinmap_find_peripheral()	335
6.2.4.13 pinmap_function()	335
6.2.4.14 pinmap_peripheral()	336
6.2.4.15 pinmap_pinout()	336
6.3 adc.h	336
6.4 drivers/comparator.c File Reference	338
6.4.1 Function Documentation	339
6.4.1.1 comparator_init()	339
6.4.1.2 comparator_read()	339
6.5 drivers/comparator.h File Reference	339
6.5.1 Detailed Description	339
6.5.2 Enumeration Type Documentation	339
6.5.2.1 ComparatorTriggerMode	339
6.5.3 Function Documentation	340
6.5.3.1 comparator_init()	340
6.5.3.2 comparator_read()	340
6.5.3.3 comparator_set_callback()	340
6.5.3.4 comparator_set_trigger()	340
6.6 comparator.h	340
6.7 drivers/gpio.c File Reference	341
6.7.1 Function Documentation	341
6.7.1.1 EXTI0_IRQHandler()	341
6.7.1.2 EXTI15_10_IRQHandler()	342
6.7.1.3 EXTI1_IRQHandler()	342
6.7.1.4 EXTI2_IRQHandler()	342
6.7.1.5 EXTI3_IRQHandler()	342
6.7.1.6 EXTI4_IRQHandler()	342
6.7.1.7 EXTI9_5_IRQHandler()	342
6.7.1.8 gpio_get()	342
6.7.1.9 gpio_get_range()	342
6.7.1.10 gpio_set()	343
6.7.1.11 gpio_set_callback()	343
6.7.1.12 gpio_set_mode()	343
6.7.1.13 gpio_set_range()	343
6.7.1.14 gpio_set_trigger()	344
6.7.1.15 gpio_toggle()	344

6.7.2 Variable Documentation	344
6.7.2.1 EXTI_port_set	344
6.7.2.2 IRQ_pin_index	344
6.7.2.3 IRQ_port_num	344
6.7.2.4 IRQ_status	344
6.7.2.5 priority	344
6.7.2.6 prioritygroup	345
6.8 drivers/gpio.h File Reference	345
6.8.1 Detailed Description	345
6.8.2 Enumeration Type Documentation	345
6.8.2.1 PinMode	345
6.8.2.2 TriggerMode	346
6.8.3 Function Documentation	346
6.8.3.1 gpio_get()	346
6.8.3.2 gpio_get_range()	346
6.8.3.3 gpio_set()	346
6.8.3.4 gpio_set_callback()	347
6.8.3.5 gpio_set_mode()	347
6.8.3.6 gpio_set_range()	347
6.8.3.7 gpio_set_trigger()	348
6.8.3.8 gpio_toggle()	348
6.9 gpio.h	348
6.10 drivers/i2c.c File Reference	349
6.10.1 Function Documentation	349
6.10.1.1 i2c_init()	349
6.10.1.2 i2c_read()	349
6.10.1.3 i2c_write()	349
6.11 drivers/i2c.h File Reference	350
6.11.1 Detailed Description	350
6.11.2 Function Documentation	350
6.11.2.1 i2c_init()	350
6.11.2.2 i2c_read()	350
6.11.2.3 i2c_write()	350
6.12 i2c.h	351
6.13 drivers/platform.h File Reference	351
6.13.1 Macro Definition Documentation	352
6.13.1.1 ADC_BITS	352
6.13.1.2 ADC_MASK	352
6.13.1.3 CLK_FREQ	352
6.13.1.4 DAC_BITS	352
6.13.1.5 DAC_MASK	352
6.13.1.6 GET_PIN_INDEX	352

6.13.1.7 GET_PORT	352
6.13.1.8 GET_PORT_INDEX	352
6.13.1.9 LED_OFF	353
6.13.1.10 LED_ON	353
6.13.1.11 P_ADC	353
6.13.1.12 P_CMP_NEG	353
6.13.1.13 P_CMP_PLUS	353
6.13.1.14 P_DBG_ISR	353
6.13.1.15 P_DBG_MAIN	353
6.13.1.16 P_IR	353
6.13.1.17 P_LCD_DATA4	353
6.13.1.18 P_LCD_DATA5	353
6.13.1.19 P_LCD_DATA6	353
6.13.1.20 P_LCD_DATA7	353
6.13.1.21 P_LCD_E	353
6.13.1.22 P_LCD_RS	353
6.13.1.23 P_LCD_RW	353
6.13.1.24 P_LED_B	354
6.13.1.25 P_LED_G	354
6.13.1.26 P_LED_R	354
6.13.1.27 P_PERIOD	354
6.13.1.28 P_RX	354
6.13.1.29 P_SAMPLE	354
6.13.1.30 P_SCL	354
6.13.1.31 P_SDA	354
6.13.1.32 P_SPEAKER	354
6.13.1.33 P_SW	354
6.13.1.34 P_SW_CR	354
6.13.1.35 P_SW_DN	354
6.13.1.36 P_SW_LT	354
6.13.1.37 P_SW_RT	354
6.13.1.38 P_SW_UP	354
6.13.1.39 P_TX	355
6.13.2 Enumeration Type Documentation	355
6.13.2.1 Pin	355
6.14 platform.h	356
6.15 drivers/stm32f4xx_adc.c File Reference	358
6.15.1 Detailed Description	360
6.16 drivers/stm32f4xx_adc.h File Reference	361
6.16.1 Detailed Description	366
6.17 stm32f4xx_adc.h	367
6.18 drivers/stm32f4xx_gpio.c File Reference	371

6.18.1 Detailed Description	372
6.19 drivers/stm32f4xx_gpio.h File Reference	373
6.19.1 Detailed Description	377
6.20 stm32f4xx_gpio.h	377
6.21 drivers/stm32f4xx_i2c.c File Reference	380
6.21.1 Detailed Description	382
6.22 drivers/stm32f4xx_i2c.h File Reference	383
6.22.1 Detailed Description	387
6.23 stm32f4xx_i2c.h	388
6.24 drivers/stm32f4xx_rcc.c File Reference	392
6.24.1 Detailed Description	395
6.24.2 Macro Definition Documentation	396
6.24.2.1 HSE_STARTUP_TIMEOUT	396
6.24.2.2 HSE_VALUE	396
6.24.2.3 HSI_VALUE	396
6.25 drivers/stm32f4xx_rcc.h File Reference	396
6.25.1 Detailed Description	402
6.26 stm32f4xx_rcc.h	402
6.27 drivers/stm32f4xx_usart.c File Reference	406
6.27.1 Detailed Description	408
6.28 drivers/stm32f4xx_usart.h File Reference	409
6.28.1 Detailed Description	412
6.29 stm32f4xx_usart.h	413
6.30 drivers/timer.c File Reference	415
6.30.1 Function Documentation	416
6.30.1.1 SysTick_Handler()	416
6.30.1.2 timer_disable()	416
6.30.1.3 timer_enable()	416
6.30.1.4 timer_init()	416
6.30.1.5 timer_set_callback()	416
6.30.2 Variable Documentation	416
6.30.2.1 timer_period	416
6.31 drivers/timer.h File Reference	417
6.31.1 Detailed Description	417
6.31.2 Function Documentation	417
6.31.2.1 timer_disable()	417
6.31.2.2 timer_enable()	417
6.31.2.3 timer_init()	417
6.31.2.4 timer_irq_handler()	417
6.31.2.5 timer_set_callback()	417
6.32 timer.h	418
6.33 drivers/uart.c File Reference	418

6.33.1 Function Documentation	418
6.33.1.1 uart_enable()	418
6.33.1.2 uart_init()	419
6.33.1.3 uart_print()	419
6.33.1.4 uart_rx()	419
6.33.1.5 uart_set_rx_callback()	419
6.33.1.6 uart_tx()	419
6.33.1.7 USART2_IRQHandler()	419
6.34 drivers/uart.h File Reference	420
6.34.1 Detailed Description	420
6.34.2 Function Documentation	420
6.34.2.1 uart_enable()	420
6.34.2.2 uart_init()	420
6.34.2.3 uart_print()	420
6.34.2.4 uart_rx()	421
6.34.2.5 uart_set_rx_callback()	421
6.34.2.6 uart_tx()	421
6.35 uart.h	421
6.36 Objects/adc.d File Reference	422
6.37 Objects/comparator.d File Reference	422
6.38 Objects/gpio.d File Reference	422
6.39 Objects/i2c.d File Reference	422
6.40 Objects/main.d File Reference	422
6.41 Objects/startup_stm32f401xe.d File Reference	422
6.42 Objects/stm32f4xx_adc.d File Reference	422
6.43 Objects/stm32f4xx_gpio.d File Reference	422
6.44 Objects/stm32f4xx_i2c.d File Reference	422
6.45 Objects/stm32f4xx_rcc.d File Reference	422
6.46 Objects/stm32f4xx_usart.d File Reference	422
6.47 Objects/system_stm32f4xx.d File Reference	422
6.48 Objects/timer.d File Reference	422
6.49 Objects/uart.d File Reference	422
6.50 RTE/_Target_1/RTE_Components.h File Reference	422
6.50.1 Macro Definition Documentation	422
6.50.1.1 CMSIS_device_header	422
6.50.1.2 RTE_DEVICE_STARTUP_STM32F4XX	422
6.51 RTE_Components.h	422
6.52 RTE/Device/STM32F401RETx/system_stm32f4xx.c File Reference	423
6.52.1 Detailed Description	423
6.53 src/main.c File Reference	424
6.53.1 Detailed Description	424
6.53.2 Function Documentation	424

6.53.2.1 main()	424
6.53.2.2 my_sqrt_fixed_point()	424
6.53.2.3 my_sqrt_int()	426

Index	427
--------------	------------

Chapter 1

Topic Index

1.1 Topics

Here is a list of all topics with brief descriptions:

STM32F4xx_StdPeriph_Driver	9
ADC	9
ADC_Private_Functions	33
Initialization and Configuration functions	34
Analog Watchdog configuration functions	37
Temperature Sensor, Vrefint (Voltage Reference internal)	40
Regular Channels Configuration functions	41
Regular Channels DMA Configuration functions	47
Injected channels Configuration functions	49
Interrupts and flags management functions	56
ADC_Exported_Constants	61
ADC_Common_mode	62
ADC_Prescaler	64
ADC_Direct_memory_access_mode_for_multi_mode	65
ADC_delay_between_2_sampling_phases	66
ADC_resolution	68
ADC_external_trigger_edge_for_regular_channels_conversion	69
ADC_extrenal_trigger_sources_for_regular_channels_conversion	70
ADC_data_align	72
ADC_channels	73
ADC_sampling_times	76
ADC_external_trigger_edge_for_injected_channels_conversion	77
ADC_extrenal_trigger_sources_for_injected_channels_conversion	78
ADC_injected_channel_selection	81
ADC_analog_watchdog_selection	82
ADC_interrupts_definition	83
ADC_flags_definition	84
ADC_thresholds	85
ADC_injected_offset	85
ADC_injected_length	85
ADC_injected_rank	86
ADC_regular_length	86
ADC_regular_rank	86
ADC_regular_discontinuous_mode_number	87
GPIO	87

GPIO_Private_Functions	99
Initialization and Configuration	99
GPIO Read and Write	102
GPIO Alternate functions configuration function	107
GPIO_Exported_Constants	110
GPIO_pins_define	110
GPIO_Pin_sources	113
GPIO_Alternat_function_selection_define	115
GPIO_Legacy	121
I2C	122
I2C_Private_Functions	140
Initialization and Configuration functions	141
Data transfers functions	149
PEC management functions	150
DMA transfers management functions	152
Interrupts events and flags management functions	153
I2C_Exported_Constants	162
I2C_mode	162
I2C_duty_cycle_in_fast_mode	163
I2C_acknowledgement	164
I2C_transfer_direction	165
I2C_acknowledged_address	165
I2C_registers	166
I2C_NACK_position	167
I2C_SMBus_alert_pin_level	168
I2C_PEC_position	168
I2C_interrupts_definition	169
I2C_flags_definition	172
I2C_Events	175
I2C_own_address1	179
I2C_clock_speed	179
RCC	179
RCC_Private_Functions	212
Internal and external clocks, PLL, CSS and MCO configuration functions	212
System AHB and APB busses clocks configuration functions	221
Peripheral clocks configuration functions	226
Interrupts and flags management functions	241
RCC_Exported_Constants	244
RCC_HSE_configuration	245
RCC_PLL_Clock_Source	245
RCC_System_Clock_Source	246
RCC_AHB_Clock_Source	247
RCC_APB1_APB2_Clock_Source	248
RCC_Interrupt_Source	249
RCC_LSE_Configuration	250
RCC_RTC_Clock_Source	250
RCC_I2S_Clock_Source	253
RCC_AHB1_Peripherals	254
RCC_AHB2_Peripherals	256
RCC_AHB3_Peripherals	257
RCC_APB1_Peripherals	257
RCC_APB2_Peripherals	259
RCC_MCO1_Clock_Source_Prescaler	261
RCC_MCO2_Clock_Source_Prescaler	262
RCC_Flag	263
USART	265
USART_Private_Functions	278
Initialization and Configuration functions	279

Data transfers functions	283
MultiProcessor Communication functions	284
LIN mode functions	285
Halfduplex mode function	287
Smartcard mode functions	288
IrDA mode functions	290
DMA transfers management functions	292
Interrupts and flags management functions	292
USART_Exported_Constants	297
USART_Word_Length	298
USART_Stop_Bits	298
USART_Parity	299
USART_Mode	299
USART_Hardware_Flow_Control	300
USART_Clock	300
USART_Clock_Polarity	301
USART_Clock_Phase	301
USART_Last_Bit	302
USART_Interrupt_definition	302
USART_Legacy	304
USART_DMA_Requests	304
USART_WakeUp_methods	305
USART_LIN_Break_Detection_Length	305
USART_IrDA_Low_Power	305
USART_Flags	306
CMSIS	307
Stm32f4xx_system	308
STM32F4xx_System_Private_Includes	308
STM32F4xx_System_Private_TypesDefinitions	308
STM32F4xx_System_Private_Defines	308
STM32F4xx_System_Private_Macros	308
STM32F4xx_System_Private_Variables	308
STM32F4xx_System_Private_FunctionPrototypes	309
STM32F4xx_System_Private_Functions	309

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

_ADC_InitTypeDef	311
ADC_ChannelConfTypeDef	312
ADC_CommonInitTypeDef	
ADC Common Init structure definition	
313	
ADC_HandleTypeDef	314
ADC_InitTypeDef	
ADC Init structure definition	
315	
analogin_s	316
GPIO_InitTypeDef	
GPIO Init structure definition	
316	
I2C_InitTypeDef	
I2C Init structure definition	
318	
PinMap	319
RCC_ClocksTypeDef	319
USART_ClockInitTypeDef	
USART Clock Init Structure definition	
320	
USART_InitTypeDef	
USART Init Structure definition	
320	

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

drivers/adc.c	323
drivers/adc.h	
Internal analogue to digital converter (ADC) controller	325
drivers/comparator.c	338
drivers/comparator.h	
Exposes functions of an internal comparator	339
drivers/gpio.c	341
drivers/gpio.h	
Implements general purpose I/O	345
drivers/i2c.c	349
drivers/i2c.h	
Controller for hardware I2C module, configured as a master	350
drivers/platform.h	351
drivers/stm32f4xx_adc.c	
This file provides firmware functions to manage the following functionalities of the Analog to Digital Converter (ADC) peripheral:	358
drivers/stm32f4xx_adc.h	
This file contains all the functions prototypes for the ADC firmware library	361
drivers/stm32f4xx_gpio.c	
This file provides firmware functions to manage the following functionalities of the GPIO peripheral:	371
drivers/stm32f4xx_gpio.h	
This file contains all the functions prototypes for the GPIO firmware library	373
drivers/stm32f4xx_i2c.c	
This file provides firmware functions to manage the following functionalities of the Inter-integrated circuit (I2C)	380
drivers/stm32f4xx_i2c.h	
This file contains all the functions prototypes for the I2C firmware library	383
drivers/stm32f4xx_rcc.c	
This file provides firmware functions to manage the following functionalities of the Reset and clock control (RCC) peripheral:	392
drivers/stm32f4xx_rcc.h	
This file contains all the functions prototypes for the RCC firmware library	396

drivers/ stm32f4xx_usart.c	
This file provides firmware functions to manage the following functionalities of the Universal synchronous asynchronous receiver transmitter (USART):	
406	
drivers/ stm32f4xx_usart.h	
This file contains all the functions prototypes for the USART firmware library	409
drivers/ timer.c	415
drivers/ timer.h	
Controller for a hardware timer module	417
drivers/ uart.c	418
drivers/ uart.h	
Controller for a hardware UART module	420
Objects/ adc.d	422
Objects/ comparator.d	422
Objects/ gpio.d	422
Objects/ i2c.d	422
Objects/ main.d	422
Objects/ startup_stm32f401xe.d	422
Objects/ stm32f4xx_adc.d	422
Objects/ stm32f4xx_gpio.d	422
Objects/ stm32f4xx_i2c.d	422
Objects/ stm32f4xx_rcc.d	422
Objects/ stm32f4xx_usart.d	422
Objects/ system_stm32f4xx.d	422
Objects/ timer.d	422
Objects/ uart.d	422
RTE/_Target_1/ RTE_Components.h	422
RTE/Device/STM32F401RETx/ system_stm32f4xx.c	
CMSIS Cortex-M4 Device Peripheral Access Layer System Source File	423
src/ main.c	
Square Root Approximation using Integer Approach - Project 1 Module 1	424

Chapter 4

Topic Documentation

4.1 STM32F4xx_StdPeriph_Driver

Modules

- [ADC](#)
ADC driver modules.
- [GPIO](#)
GPIO driver modules.
- [I2C](#)
I2C driver modules.
- [RCC](#)
RCC driver modules.
- [USART](#)
USART driver modules.

4.1.1 Detailed Description

4.1.2 ADC

ADC driver modules.

Modules

- [ADC_Private_Functions](#)
- [ADC_Exported_Constants](#)

Data Structures

- struct [ADC_InitTypeDef](#)
ADC Init structure definition
- struct [ADC_CommonInitTypeDef](#)
ADC Common Init structure definition

Macros

- #define [CR1_DISCNUM_RESET](#) ((uint32_t)0xFFFF1FFF)
- #define [CR1_AWDCH_RESET](#) ((uint32_t)0xFFFFFFE0)
- #define [CR1_AWDMode_RESET](#) ((uint32_t)0xFF3FFDFF)
- #define [CR1_CLEAR_MASK](#) ((uint32_t)0xFCFFFEFF)
- #define [CR2_EXTEN_RESET](#) ((uint32_t)0xCFFFFFFF)
- #define [CR2_JEXTEN_RESET](#) ((uint32_t)0xFFCFFFFFFF)
- #define [CR2_JEXTSEL_RESET](#) ((uint32_t)0xFFF0FFFF)
- #define [CR2_CLEAR_MASK](#) ((uint32_t)0xC0FFF7FD)
- #define [SQR3_SQ_SET](#) ((uint32_t)0x0000001F)
- #define [SQR2_SQ_SET](#) ((uint32_t)0x0000001F)
- #define [SQR1_SQ_SET](#) ((uint32_t)0x0000001F)
- #define [SQR1_L_RESET](#) ((uint32_t)0xFF0FFFFF)
- #define [JSQR_JSQ_SET](#) ((uint32_t)0x0000001F)
- #define [JSQR_JL_SET](#) ((uint32_t)0x00300000)
- #define [JSQR_JL_RESET](#) ((uint32_t)0xFFCFFFFFFF)
- #define [SMPR1_SMP_SET](#) ((uint32_t)0x00000007)
- #define [SMPR2_SMP_SET](#) ((uint32_t)0x00000007)
- #define [JDR_OFFSET](#) ((uint8_t)0x28)
- #define [CDR_ADDRESS](#) ((uint32_t)0x40012308)
- #define [CR_CLEAR_MASK](#) ((uint32_t)0xFFFC30E0)

Functions

- void [ADC_DeInit](#) (void)
Deinitializes all ADCs peripherals registers to their default reset values.
- void [ADC_Init](#) (ADC_TypeDef *ADCx, [ADC_InitTypeDef](#) *ADC_InitStruct)
Initializes the ADCx peripheral according to the specified parameters in the ADC_InitStruct.
- void [ADC_StructInit](#) ([ADC_InitTypeDef](#) *ADC_InitStruct)
Fills each ADC_InitStruct member with its default value.
- void [ADC_CommonInit](#) ([ADC_CommonInitTypeDef](#) *ADC_CommonInitStruct)
Initializes the ADCs peripherals according to the specified parameters in the ADC_CommonInitStruct.
- void [ADC_CommonStructInit](#) ([ADC_CommonInitTypeDef](#) *ADC_CommonInitStruct)
Fills each ADC_CommonInitStruct member with its default value.
- void [ADC_Cmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the specified ADC peripheral.
- void [ADC_AnalogWatchdogCmd](#) (ADC_TypeDef *ADCx, uint32_t ADC_AnalogWatchdog)
Enables or disables the analog watchdog on single/all regular or injected channels.
- void [ADC_AnalogWatchdogThresholdsConfig](#) (ADC_TypeDef *ADCx, uint16_t HighThreshold, uint16_t LowThreshold)
Configures the high and low thresholds of the analog watchdog.
- void [ADC_AnalogWatchdogSingleChannelConfig](#) (ADC_TypeDef *ADCx, uint8_t ADC_Channel)
Configures the analog watchdog guarded single channel.
- void [ADC_TempSensorVrefintCmd](#) (FunctionalState NewState)
Enables or disables the temperature sensor and Vrefint channels.
- void [ADC_VBATCmd](#) (FunctionalState NewState)
Enables or disables the VBAT (Voltage Battery) channel.
- void [ADC-RegularChannelConfig](#) (ADC_TypeDef *ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)
Configures for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
- void [ADC_SoftwareStartConv](#) (ADC_TypeDef *ADCx)

- Enables the selected ADC software start conversion of the regular channels.*

 - FlagStatus [ADC_GetSoftwareStartConvStatus](#) (ADC_TypeDef *ADCx)

Gets the selected ADC Software start regular conversion Status.
- void [ADC_EOCOnEachRegularChannelCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the EOC on each regular channel conversion.
- void [ADC_ContinuousModeCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the ADC continuous conversion mode.
- void [ADC_DiscModeChannelCountConfig](#) (ADC_TypeDef *ADCx, uint8_t Number)

Configures the discontinuous mode for the selected ADC regular group channel.
- void [ADC_DiscModeCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the discontinuous mode on regular group channel for the specified ADC.
- uint16_t [ADC_GetConversionValue](#) (ADC_TypeDef *ADCx)

Returns the last ADCx conversion result data for regular channel.
- uint32_t [ADC_GetMultiModeConversionValue](#) (void)

Returns the last ADC1, ADC2 and ADC3 regular conversions results data in the selected multi mode.
- void [ADC_DMAMCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the specified ADC DMA request.
- void [ADC_DMAResultRequestAfterLastTransferCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the ADC DMA request after last transfer (Single-ADC mode)
- void [ADC_MultiModeDMAResultRequestAfterLastTransferCmd](#) (FunctionalState NewState)

Enables or disables the ADC DMA request after last transfer in multi ADC mode
- void [ADC_InjectedChannelConfig](#) (ADC_TypeDef *ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)

Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time.
- void [ADC_InjectedSequencerLengthConfig](#) (ADC_TypeDef *ADCx, uint8_t Length)

Configures the sequencer length for injected channels.
- void [ADC_SetInjectedOffset](#) (ADC_TypeDef *ADCx, uint8_t ADC_InjectedChannel, uint16_t Offset)

Set the injected channels conversion value offset.
- void [ADC_ExternalTrigInjectedConvConfig](#) (ADC_TypeDef *ADCx, uint32_t ADC_ExternalTrigInjecConv)

Configures the ADCx external trigger for injected channels conversion.
- void [ADC_ExternalTrigInjectedConvEdgeConfig](#) (ADC_TypeDef *ADCx, uint32_t ADC_ExternalTrigInjecConvEdge)

Configures the ADCx external trigger edge for injected channels conversion.
- void [ADC_SoftwareStartInjectedConv](#) (ADC_TypeDef *ADCx)

Enables the selected ADC software start conversion of the injected channels.
- FlagStatus [ADC_GetSoftwareStartInjectedConvCmdStatus](#) (ADC_TypeDef *ADCx)

Gets the selected ADC Software start injected conversion Status.
- void [ADC_AutoInjectedConvCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the selected ADC automatic injected group conversion after regular one.
- void [ADC_InjectedDiscModeCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the discontinuous mode for injected group channel for the specified ADC.
- uint16_t [ADC_GetInjectedConversionValue](#) (ADC_TypeDef *ADCx, uint8_t ADC_InjectedChannel)

Returns the ADC injected channel conversion result.
- void [ADC_ITConfig](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT, FunctionalState NewState)

Enables or disables the specified ADC interrupts.
- FlagStatus [ADC_GetFlagStatus](#) (ADC_TypeDef *ADCx, uint8_t ADC_FLAG)

Checks whether the specified ADC flag is set or not.
- void [ADC_ClearFlag](#) (ADC_TypeDef *ADCx, uint8_t ADC_FLAG)

Clears the ADCx's pending flags.

- ITStatus [ADC_GetITStatus](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT)
Checks whether the specified ADC interrupt has occurred or not.
- void [ADC_ClearITPendingBit](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT)
Clears the ADCx's interrupt pending bits.

4.1.2.1 Detailed Description

ADC driver modules.

4.1.2.2 Macro Definition Documentation

4.1.2.2.1 CDR_ADDRESS

```
#define CDR_ADDRESS ((uint32_t)0x40012308)
```

4.1.2.2.2 CR1_AWDCH_RESET

```
#define CR1_AWDCH_RESET ((uint32_t)0xFFFFFE0)
```

4.1.2.2.3 CR1_AWDMode_RESET

```
#define CR1_AWDMode_RESET ((uint32_t)0xFF3FFDFF)
```

4.1.2.2.4 CR1_CLEAR_MASK

```
#define CR1_CLEAR_MASK ((uint32_t)0xFCFFFFFF)
```

4.1.2.2.5 CR1_DISCNUM_RESET

```
#define CR1_DISCNUM_RESET ((uint32_t)0xFFFF1FFF)
```

4.1.2.2.6 CR2_CLEAR_MASK

```
#define CR2_CLEAR_MASK ((uint32_t)0xC0FFF7FD)
```

4.1.2.2.7 CR2_EXTEN_RESET

```
#define CR2_EXTEN_RESET ((uint32_t)0xCFFFFFFF)
```

4.1.2.2.8 CR2_JEXTEN_RESET

```
#define CR2_JEXTEN_RESET ((uint32_t)0xFFCFFFFFF)
```

4.1.2.2.9 CR2_JEXTSEL_RESET

```
#define CR2_JEXTSEL_RESET ((uint32_t)0xFFFF0FFF)
```

4.1.2.2.10 CR_CLEAR_MASK

```
#define CR_CLEAR_MASK ((uint32_t)0xFFFC30E0)
```

4.1.2.2.11 JDR_OFFSET

```
#define JDR_OFFSET ((uint8_t)0x28)
```

4.1.2.2.12 JSQR_JL_RESET

```
#define JSQR_JL_RESET ((uint32_t)0xFFCFFFFF)
```

4.1.2.2.13 JSQR_JL_SET

```
#define JSQR_JL_SET ((uint32_t)0x00300000)
```

4.1.2.2.14 JSQR_JSQ_SET

```
#define JSQR_JSQ_SET ((uint32_t)0x0000001F)
```

4.1.2.2.15 SMPR1_SMP_SET

```
#define SMPR1_SMP_SET ((uint32_t)0x00000007)
```

4.1.2.2.16 SMPR2_SMP_SET

```
#define SMPR2_SMP_SET ((uint32_t)0x00000007)
```

4.1.2.2.17 SQR1_L_RESET

```
#define SQR1_L_RESET ((uint32_t)0xFF0FFFFF)
```

4.1.2.2.18 SQR1_SQ_SET

```
#define SQR1_SQ_SET ((uint32_t)0x0000001F)
```

4.1.2.2.19 SQR2_SQ_SET

```
#define SQR2_SQ_SET ((uint32_t)0x0000001F)
```

4.1.2.2.20 SQR3_SQ_SET

```
#define SQR3_SQ_SET ((uint32_t)0x0000001F)
```

4.1.2.3 Function Documentation

4.1.2.3.1 ADC_AnalogWatchdogCmd()

```
void ADC_AnalogWatchdogCmd (
    ADC_TypeDef * ADCx,
    uint32_t ADC_AnalogWatchdog )
```

Enables or disables the analog watchdog on single/all regular or injected channels.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_AnalogWatchdog</i>	<p>the ADC analog watchdog configuration. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>ADC_AnalogWatchdog_SingleRegEnable</code>: Analog watchdog on a single regular channel • <code>ADC_AnalogWatchdog_SingleInjecEnable</code>: Analog watchdog on a single injected channel • <code>ADC_AnalogWatchdog_SingleRegOrInjecEnable</code>: Analog watchdog on a single regular or injected channel • <code>ADC_AnalogWatchdog_AllRegEnable</code>: Analog watchdog on all regular channel • <code>ADC_AnalogWatchdog_AllInjecEnable</code>: Analog watchdog on all injected channel • <code>ADC_AnalogWatchdog_AllRegAllInjecEnable</code>: Analog watchdog on all regular and injected channels • <code>ADC_AnalogWatchdog_None</code>: No channel guarded by the analog watchdog

Return values

<i>None</i>	
-------------	--

4.1.2.3.2 ADC_AnalogWatchdogSingleChannelConfig()

```
void ADC_AnalogWatchdogSingleChannelConfig (
```

```
ADC_TypeDef * ADCx,
uint8_t ADC_Channel )
```

Configures the analog watchdog guarded single channel.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_Channel</i>	the ADC channel to configure for the analog watchdog. This parameter can be one of the following values: <ul style="list-style-type: none"> • ADC_Channel_0: ADC Channel0 selected • ADC_Channel_1: ADC Channel1 selected • ADC_Channel_2: ADC Channel2 selected • ADC_Channel_3: ADC Channel3 selected • ADC_Channel_4: ADC Channel4 selected • ADC_Channel_5: ADC Channel5 selected • ADC_Channel_6: ADC Channel6 selected • ADC_Channel_7: ADC Channel7 selected • ADC_Channel_8: ADC Channel8 selected • ADC_Channel_9: ADC Channel9 selected • ADC_Channel_10: ADC Channel10 selected • ADC_Channel_11: ADC Channel11 selected • ADC_Channel_12: ADC Channel12 selected • ADC_Channel_13: ADC Channel13 selected • ADC_Channel_14: ADC Channel14 selected • ADC_Channel_15: ADC Channel15 selected • ADC_Channel_16: ADC Channel16 selected • ADC_Channel_17: ADC Channel17 selected • ADC_Channel_18: ADC Channel18 selected

Return values

<i>None</i>	
-------------	--

4.1.2.3.3 ADC_AnalogWatchdogThresholdsConfig()

```
void ADC_AnalogWatchdogThresholdsConfig (
    ADC_TypeDef * ADCx,
    uint16_t HighThreshold,
    uint16_t LowThreshold )
```

Configures the high and low thresholds of the analog watchdog.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>HighThreshold</i>	the ADC analog watchdog High threshold value. This parameter must be a 12-bit value.
<i>LowThreshold</i>	the ADC analog watchdog Low threshold value. This parameter must be a 12-bit value.

Return values

<i>None</i>	
-------------	--

4.1.2.3.4 ADC_AutoInjectedConvCmd()

```
void ADC_AutoInjectedConvCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the selected ADC automatic injected group conversion after regular one.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC auto injected conversion This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.2.3.5 ADC_ClearFlag()

```
void ADC_ClearFlag (
    ADC_TypeDef * ADCx,
    uint8_t ADC_FLAG )
```

Clears the ADCx's pending flags.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_FLAG</i>	specifies the flag to clear. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • ADC_FLAG_AWD: Analog watchdog flag • ADC_FLAG_EOC: End of conversion flag • ADC_FLAG_JEOC: End of injected group conversion flag • ADC_FLAG_JSTRT: Start of injected group conversion flag • ADC_FLAG_STRT: Start of regular group conversion flag • ADC_FLAG_OVR: Overrun flag

Return values

<i>None</i>	
-------------	--

4.1.2.3.6 ADC_ClearITPendingBit()

```
void ADC_ClearITPendingBit (
    ADC_TypeDef * ADCx,
    uint16_t ADC_IT )
```

Clears the ADCx's interrupt pending bits.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADCx</i> ↔ <i>_IT</i>	specifies the ADC interrupt pending bit to clear. This parameter can be one of the following values: <ul style="list-style-type: none"> • ADC_IT_EOC: End of conversion interrupt mask • ADC_IT_AWD: Analog watchdog interrupt mask • ADC_IT_JEOC: End of injected conversion interrupt mask • ADC_IT_OVR: Overrun interrupt mask

Return values

<i>None</i>	
-------------	--

4.1.2.3.7 ADC_Cmd()

```
void ADC_Cmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the specified ADC peripheral.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the ADCx peripheral. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.2.3.8 ADC_CommonInit()

```
void ADC_CommonInit (
    ADC_CommonInitTypeDef * ADC_CommonInitStruct )
```

Initializes the ADCs peripherals according to the specified parameters in the ADC_CommonInitStruct.

Parameters

<i>ADC_CommonInitStruct</i>	pointer to an ADC_CommonInitTypeDef structure that contains the configuration information for All ADCs peripherals.
-----------------------------	---

Return values

<i>None</i>	
-------------	--

4.1.2.3.9 ADC_CommonStructInit()

```
void ADC_CommonStructInit (
    ADC_CommonInitTypeDef * ADC_CommonInitStruct )
```

Fills each ADC_CommonInitStruct member with its default value.

Parameters

<i>ADC_CommonInitStruct</i>	pointer to an ADC_CommonInitTypeDef structure which will be initialized.
-----------------------------	--

Return values

<i>None</i>	
-------------	--

4.1.2.3.10 ADC_ContinuousModeCmd()

```
void ADC_ContinuousModeCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the ADC continuous conversion mode.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC continuous conversion mode This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.2.3.11 ADC_DeInit()

```
void ADC_DeInit (
    void )
```

Deinitializes all ADCs peripherals registers to their default reset values.

Parameters

<i>None</i>	
-------------	--

Return values

<i>None</i>	
-------------	--

4.1.2.3.12 ADC_DiscModeChannelCountConfig()

```
void ADC_DiscModeChannelCountConfig (
    ADC_TypeDef * ADCx,
    uint8_t Number )
```

Configures the discontinuous mode for the selected ADC regular group channel.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>Number</i>	specifies the discontinuous mode regular channel count value. This number must be between 1 and 8.

Return values

<i>None</i>	
-------------	--

4.1.2.3.13 ADC_DiscModeCmd()

```
void ADC_DiscModeCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the discontinuous mode on regular group channel for the specified ADC.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC discontinuous mode on regular group channel. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.2.3.14 ADC_DMACmd()

```
void ADC_DMACmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the specified ADC DMA request.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC DMA transfer. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.2.3.15 ADC_DMAResultAfterLastTransferCmd()

```
void ADC_DMAResultAfterLastTransferCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the ADC DMA request after last transfer (Single-ADC mode)

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC DMA request after last transfer. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.2.3.16 ADC_EOCOnEachRegularChannelCmd()

```
void ADC_EOCOnEachRegularChannelCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the EOC on each regular channel conversion.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC EOC flag rising This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.2.3.17 ADC_ExternalTrigInjectedConvConfig()

```
void ADC_ExternalTrigInjectedConvConfig (
    ADC_TypeDef * ADCx,
    uint32_t ADC_ExternalTrigInjecConv )
```

Configures the ADCx external trigger for injected channels conversion.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Parameters

<i>ADC_ExternalTrigInjecConv</i>	<p>specifies the ADC trigger to start injected conversion. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <i>ADC_ExternalTrigInjecConv_T1_CC4</i>: Timer1 capture compare4 selected • <i>ADC_ExternalTrigInjecConv_T1_TRGO</i>: Timer1 TRGO event selected • <i>ADC_ExternalTrigInjecConv_T2_CC1</i>: Timer2 capture compare1 selected • <i>ADC_ExternalTrigInjecConv_T2_TRGO</i>: Timer2 TRGO event selected • <i>ADC_ExternalTrigInjecConv_T3_CC2</i>: Timer3 capture compare2 selected • <i>ADC_ExternalTrigInjecConv_T3_CC4</i>: Timer3 capture compare4 selected • <i>ADC_ExternalTrigInjecConv_T4_CC1</i>: Timer4 capture compare1 selected • <i>ADC_ExternalTrigInjecConv_T4_CC2</i>: Timer4 capture compare2 selected • <i>ADC_ExternalTrigInjecConv_T4_CC3</i>: Timer4 capture compare3 selected • <i>ADC_ExternalTrigInjecConv_T4_TRGO</i>: Timer4 TRGO event selected • <i>ADC_ExternalTrigInjecConv_T5_CC4</i>: Timer5 capture compare4 selected • <i>ADC_ExternalTrigInjecConv_T5_TRGO</i>: Timer5 TRGO event selected • <i>ADC_ExternalTrigInjecConv_T8_CC2</i>: Timer8 capture compare2 selected • <i>ADC_ExternalTrigInjecConv_T8_CC3</i>: Timer8 capture compare3 selected • <i>ADC_ExternalTrigInjecConv_T8_CC4</i>: Timer8 capture compare4 selected • <i>ADC_ExternalTrigInjecConv_Ext_IT15</i>: External interrupt line 15 event selected
----------------------------------	---

Return values

<i>None</i>	
-------------	--

4.1.2.3.18 ADC_ExternalTrigInjectedConvEdgeConfig()

```
void ADC_ExternalTrigInjectedConvEdgeConfig (
    ADC_TypeDef * ADCx,
    uint32_t ADC_ExternalTrigInjecConvEdge )
```

Configures the ADCx external trigger edge for injected channels conversion.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_ExternalTrigInjecConvEdge</i>	<p>specifies the ADC external trigger edge to start injected conversion. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> ADC_ExternalTrigInjecConvEdge_None: external trigger disabled for injected conversion ADC_ExternalTrigInjecConvEdge_Rising: detection on rising edge ADC_ExternalTrigInjecConvEdge_Falling: detection on falling edge ADC_ExternalTrigInjecConvEdge_RisingFalling: detection on both rising and falling edge

Return values

<i>None</i>	
-------------	--

4.1.2.3.19 ADC_GetConversionValue()

```
uint16_t ADC_GetConversionValue (
    ADC_TypeDef * ADCx )
```

Returns the last ADCx conversion result data for regular channel.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Return values

<i>The</i>	Data conversion value.
------------	------------------------

4.1.2.3.20 ADC_GetFlagStatus()

```
FlagStatus ADC_GetFlagStatus (
    ADC_TypeDef * ADCx,
    uint8_t ADC_FLAG )
```

Checks whether the specified ADC flag is set or not.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_FLAG</i>	specifies the flag to check. This parameter can be one of the following values: <ul style="list-style-type: none"> • <code>ADC_FLAG_AWD</code>: Analog watchdog flag • <code>ADC_FLAG_EOC</code>: End of conversion flag • <code>ADC_FLAG_JEOC</code>: End of injected group conversion flag • <code>ADC_FLAG_JSTRT</code>: Start of injected group conversion flag • <code>ADC_FLAG_STRT</code>: Start of regular group conversion flag • <code>ADC_FLAG_OVR</code>: Overrun flag

Return values

<i>The</i>	new state of <code>ADC_FLAG</code> (SET or RESET).
------------	--

4.1.2.3.21 ADC_GetInjectedConversionValue()

```
uint16_t ADC_GetInjectedConversionValue (
    ADC_TypeDef * ADCx,
    uint8_t ADC_InjectedChannel )
```

Returns the ADC injected channel conversion result.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_InjectedChannel</i>	the converted ADC injected channel. This parameter can be one of the following values: <ul style="list-style-type: none"> • <code>ADC_InjectedChannel_1</code>: Injected Channel1 selected • <code>ADC_InjectedChannel_2</code>: Injected Channel2 selected • <code>ADC_InjectedChannel_3</code>: Injected Channel3 selected • <code>ADC_InjectedChannel_4</code>: Injected Channel4 selected

Return values

<i>The</i>	Data conversion value.
------------	------------------------

4.1.2.3.22 ADC_GetITStatus()

```
ITStatus ADC_GetITStatus (
    ADC_TypeDef * ADCx,
    uint16_t ADC_IT )
```

Checks whether the specified ADC interrupt has occurred or not.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_IT</i>	specifies the ADC interrupt source to check. This parameter can be one of the following values: <ul style="list-style-type: none"> • ADC_IT_EOC: End of conversion interrupt mask • ADC_IT_AWD: Analog watchdog interrupt mask • ADC_IT_JEOC: End of injected conversion interrupt mask • ADC_IT_OVR: Overrun interrupt mask

Return values

<i>The</i>	new state of ADC_IT (SET or RESET).
------------	-------------------------------------

4.1.2.3.23 ADC_GetMultiModeConversionValue()

```
uint32_t ADC_GetMultiModeConversionValue (
    void )
```

Returns the last ADC1, ADC2 and ADC3 regular conversions results data in the selected multi mode.

Parameters

<i>None</i>	
-------------	--

Return values

<i>The</i>	Data conversion value.
------------	------------------------

Note

In dual mode, the value returned by this function is as following Data[15:0] : these bits contain the regular data of ADC1. Data[31:16]: these bits contain the regular data of ADC2.

In triple mode, the value returned by this function is as following Data[15:0] : these bits contain alternatively the regular data of ADC1, ADC3 and ADC2. Data[31:16]: these bits contain alternatively the regular data of ADC2, ADC1 and ADC3.

4.1.2.3.24 ADC_GetSoftwareStartConvStatus()

```
FlagStatus ADC_GetSoftwareStartConvStatus (
    ADC_TypeDef * ADCx )
```

Gets the selected ADC Software start regular conversion Status.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Return values

<i>The</i>	new state of ADC software start conversion (SET or RESET).
------------	--

4.1.2.3.25 ADC_GetSoftwareStartInjectedConvCmdStatus()

```
FlagStatus ADC_GetSoftwareStartInjectedConvCmdStatus (
    ADC_TypeDef * ADCx )
```

Gets the selected ADC Software start injected conversion Status.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Return values

<i>The</i>	new state of ADC software start injected conversion (SET or RESET).
------------	---

4.1.2.3.26 ADC_Init()

```
void ADC_Init (
    ADC_TypeDef * ADCx,
    ADC_InitTypeDef * ADC_InitStruct )
```

Initializes the ADCx peripheral according to the specified parameters in the ADC_InitStruct.

Note

This function is used to configure the global features of the ADC (Resolution and Data Alignment), however, the rest of the configuration parameters are specific to the regular channels group (scan mode activation, continuous mode activation, External trigger source and edge, number of conversion in the regular channels group sequencer).

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_InitStruct</i>	pointer to an ADC_InitTypeDef structure that contains the configuration information for the specified ADC peripheral.

Return values

Return values

None	
------	--

4.1.2.3.27 ADC_InjectedChannelConfig()

```
void ADC_InjectedChannelConfig (
    ADC_TypeDef * ADCx,
    uint8_t ADC_Channel,
    uint8_t Rank,
    uint8_t ADC_SampleTime )
```

Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_Channel</i>	the ADC channel to configure. This parameter can be one of the following values: <ul style="list-style-type: none"> • ADC_Channel_0: ADC Channel0 selected • ADC_Channel_1: ADC Channel1 selected • ADC_Channel_2: ADC Channel2 selected • ADC_Channel_3: ADC Channel3 selected • ADC_Channel_4: ADC Channel4 selected • ADC_Channel_5: ADC Channel5 selected • ADC_Channel_6: ADC Channel6 selected • ADC_Channel_7: ADC Channel7 selected • ADC_Channel_8: ADC Channel8 selected • ADC_Channel_9: ADC Channel9 selected • ADC_Channel_10: ADC Channel10 selected • ADC_Channel_11: ADC Channel11 selected • ADC_Channel_12: ADC Channel12 selected • ADC_Channel_13: ADC Channel13 selected • ADC_Channel_14: ADC Channel14 selected • ADC_Channel_15: ADC Channel15 selected • ADC_Channel_16: ADC Channel16 selected • ADC_Channel_17: ADC Channel17 selected • ADC_Channel_18: ADC Channel18 selected
<i>Rank</i>	The rank in the injected group sequencer. This parameter must be between 1 to 4.

Parameters

<i>ADC_SampleTime</i>	<p>The sample time value to be set for the selected channel. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>ADC_SampleTime_3Cycles</code>: Sample time equal to 3 cycles • <code>ADC_SampleTime_15Cycles</code>: Sample time equal to 15 cycles • <code>ADC_SampleTime_28Cycles</code>: Sample time equal to 28 cycles • <code>ADC_SampleTime_56Cycles</code>: Sample time equal to 56 cycles • <code>ADC_SampleTime_84Cycles</code>: Sample time equal to 84 cycles • <code>ADC_SampleTime_112Cycles</code>: Sample time equal to 112 cycles • <code>ADC_SampleTime_144Cycles</code>: Sample time equal to 144 cycles • <code>ADC_SampleTime_480Cycles</code>: Sample time equal to 480 cycles
-----------------------	--

Return values

<i>None</i>	
-------------	--

4.1.2.3.28 ADC_InjectedDiscModeCmd()

```
void ADC_InjectedDiscModeCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the discontinuous mode for injected group channel for the specified ADC.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC discontinuous mode on injected group channel. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .

Return values

<i>None</i>	
-------------	--

4.1.2.3.29 ADC_InjectedSequencerLengthConfig()

```
void ADC_InjectedSequencerLengthConfig (
    ADC_TypeDef * ADCx,
    uint8_t Length )
```

Configures the sequencer length for injected channels.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>Length</i>	The sequencer length. This parameter must be a number between 1 to 4.

Return values

<i>None</i>	
-------------	--

4.1.2.3.30 ADC_ITConfig()

```
void ADC_ITConfig (
    ADC_TypeDef * ADCx,
    uint16_t ADC_IT,
    FunctionalState NewState )
```

Enables or disables the specified ADC interrupts.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_IT</i>	specifies the ADC interrupt sources to be enabled or disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> • ADC_IT_EOC: End of conversion interrupt mask • ADC_IT_AWD: Analog watchdog interrupt mask • ADC_IT_JEOC: End of injected conversion interrupt mask • ADC_IT_OVR: Overrun interrupt enable
<i>NewState</i>	new state of the specified ADC interrupts. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.2.3.31 ADC_MultiModeDMARequestAfterLastTransferCmd()

```
void ADC_MultiModeDMARequestAfterLastTransferCmd (
    FunctionalState NewState )
```

Enables or disables the ADC DMA request after last transfer in multi ADC mode

Parameters

<i>NewState</i>	new state of the selected ADC DMA request after last transfer. This parameter can be: ENABLE or DISABLE.
-----------------	--

Note

if Enabled, DMA requests are issued as long as data are converted and DMA mode for multi ADC mode (selected using [ADC_CommonInit\(\)](#) function by ADC_CommonInitStruct.ADC_DMAAccessMode structure member) is ADC_DMAAccessMode_1, ADC_DMAAccessMode_2 or ADC_DMAAccessMode_3.

Return values

<i>None</i>	
-------------	--

4.1.2.3.32 ADC_RegularChannelConfig()

```
void ADC_RegularChannelConfig (
    ADC_TypeDef * ADCx,
    uint8_t ADC_Channel,
    uint8_t Rank,
    uint8_t ADC_SampleTime )
```

Configures for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_Channel</i>	<p>the ADC channel to configure. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • ADC_Channel_0: ADC Channel0 selected • ADC_Channel_1: ADC Channel1 selected • ADC_Channel_2: ADC Channel2 selected • ADC_Channel_3: ADC Channel3 selected • ADC_Channel_4: ADC Channel4 selected • ADC_Channel_5: ADC Channel5 selected • ADC_Channel_6: ADC Channel6 selected • ADC_Channel_7: ADC Channel7 selected • ADC_Channel_8: ADC Channel8 selected • ADC_Channel_9: ADC Channel9 selected • ADC_Channel_10: ADC Channel10 selected • ADC_Channel_11: ADC Channel11 selected • ADC_Channel_12: ADC Channel12 selected • ADC_Channel_13: ADC Channel13 selected • ADC_Channel_14: ADC Channel14 selected • ADC_Channel_15: ADC Channel15 selected • ADC_Channel_16: ADC Channel16 selected • ADC_Channel_17: ADC Channel17 selected • ADC_Channel_18: ADC Channel18 selected
	Generated by Doxygen

Parameters

<i>Rank</i>	The rank in the regular group sequencer. This parameter must be between 1 to 16.
<i>ADC_SampleTime</i>	<p>The sample time value to be set for the selected channel. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>ADC_SampleTime_3Cycles</code>: Sample time equal to 3 cycles • <code>ADC_SampleTime_15Cycles</code>: Sample time equal to 15 cycles • <code>ADC_SampleTime_28Cycles</code>: Sample time equal to 28 cycles • <code>ADC_SampleTime_56Cycles</code>: Sample time equal to 56 cycles • <code>ADC_SampleTime_84Cycles</code>: Sample time equal to 84 cycles • <code>ADC_SampleTime_112Cycles</code>: Sample time equal to 112 cycles • <code>ADC_SampleTime_144Cycles</code>: Sample time equal to 144 cycles • <code>ADC_SampleTime_480Cycles</code>: Sample time equal to 480 cycles

Return values

<i>None</i>	
-------------	--

4.1.2.3.33 ADC_SetInjectedOffset()

```
void ADC_SetInjectedOffset (
    ADC_TypeDef * ADCx,
    uint8_t ADC_InjectedChannel,
    uint16_t Offset )
```

Set the injected channels conversion value offset.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_InjectedChannel</i>	<p>the ADC injected channel to set its offset. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>ADC_InjectedChannel_1</code>: Injected Channel1 selected • <code>ADC_InjectedChannel_2</code>: Injected Channel2 selected • <code>ADC_InjectedChannel_3</code>: Injected Channel3 selected • <code>ADC_InjectedChannel_4</code>: Injected Channel4 selected
<i>Offset</i>	the offset value for the selected ADC injected channel This parameter must be a 12bit value.

Return values

<i>None</i>	
-------------	--

4.1.2.3.34 ADC_SoftwareStartConv()

```
void ADC_SoftwareStartConv (
    ADC_TypeDef * ADCx )
```

Enables the selected ADC software start conversion of the regular channels.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Return values

<i>None</i>	
-------------	--

4.1.2.3.35 ADC_SoftwareStartInjectedConv()

```
void ADC_SoftwareStartInjectedConv (
    ADC_TypeDef * ADCx )
```

Enables the selected ADC software start conversion of the injected channels.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Return values

<i>None</i>	
-------------	--

4.1.2.3.36 ADC_StructInit()

```
void ADC_StructInit (
    ADC_InitTypeDef * ADC_InitStruct )
```

Fills each ADC_InitStruct member with its default value.

Note

This function is used to initialize the global features of the ADC (Resolution and Data Alignment), however, the rest of the configuration parameters are specific to the regular channels group (scan mode activation, continuous mode activation, External trigger source and edge, number of conversion in the regular channels group sequencer).

Parameters

<i>ADC_InitStruct</i>	pointer to an ADC_InitTypeDef structure which will be initialized.
-----------------------	--

Return values

<i>None</i>	
-------------	--

4.1.2.3.37 ADC_TempSensorVrefintCmd()

```
void ADC_TempSensorVrefintCmd (
    FunctionalState NewState )
```

Enables or disables the temperature sensor and Vrefint channels.

Parameters

<i>NewState</i>	new state of the temperature sensor and Vrefint channels. This parameter can be: ENABLE or DISABLE.
-----------------	---

Return values

<i>None</i>	
-------------	--

4.1.2.3.38 ADC_VBATCmd()

```
void ADC_VBATCmd (
    FunctionalState NewState )
```

Enables or disables the VBAT (Voltage Battery) channel.

Parameters

<i>NewState</i>	new state of the VBAT channel. This parameter can be: ENABLE or DISABLE.
-----------------	--

Return values

<i>None</i>	
-------------	--

4.1.2.4 ADC_Private_Functions

Modules

- [Initialization and Configuration functions](#)
Initialization and Configuration functions.

- [Analog Watchdog configuration functions](#)
Analog Watchdog configuration functions.
- [Temperature Sensor, Vrefint \(Voltage Reference internal\)](#)
Temperature Sensor, Vrefint and VBAT management functions.
- [Regular Channels Configuration functions](#)
Regular Channels Configuration functions.
- [Regular Channels DMA Configuration functions](#)
Regular Channels DMA Configuration functions.
- [Injected channels Configuration functions](#)
Injected channels Configuration functions.
- [Interrupts and flags management functions](#)
Interrupts and flags management functions.

4.1.2.4.1 Detailed Description

4.1.2.4.2 Initialization and Configuration functions

Initialization and Configuration functions.

Functions

- void [ADC_DeInit](#) (void)
Deinitializes all ADCs peripherals registers to their default reset values.
- void [ADC_Init](#) (ADC_TypeDef *ADCx, [ADC_InitTypeDef](#) *ADC_InitStruct)
Initializes the ADCx peripheral according to the specified parameters in the ADC_InitStruct.
- void [ADC_StructInit](#) ([ADC_InitTypeDef](#) *ADC_InitStruct)
Fills each ADC_InitStruct member with its default value.
- void [ADC_CommonInit](#) ([ADC_CommonInitTypeDef](#) *ADC_CommonInitStruct)
Initializes the ADCs peripherals according to the specified parameters in the ADC_CommonInitStruct.
- void [ADC_CommonStructInit](#) ([ADC_CommonInitTypeDef](#) *ADC_CommonInitStruct)
Fills each ADC_CommonInitStruct member with its default value.
- void [ADC_Cmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the specified ADC peripheral.

4.1.2.4.2.1 Detailed Description

Initialization and Configuration functions.

```
=====
                          Initialization and Configuration functions
=====

This section provides functions allowing to:
- Initialize and configure the ADC Prescaler
- ADC Conversion Resolution (12bit..6bit)
- Scan Conversion Mode (multichannels or one channel) for regular group
- ADC Continuous Conversion Mode (Continuous or Single conversion) for
  regular group
- External trigger Edge and source of regular group,
- Converted data alignment (left or right)
- The number of ADC conversions that will be done using the sequencer for
  regular channel group
- Multi ADC mode selection
- Direct memory access mode selection for multi ADC mode
- Delay between 2 sampling phases (used in dual or triple interleaved modes)
- Enable or disable the ADC peripheral
```


4.1.2.4.2.2 Function Documentation

ADC_Cmd()

```
void ADC_Cmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the specified ADC peripheral.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the ADCx peripheral. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

ADC_CommonInit()

```
void ADC_CommonInit (
    ADC_CommonInitTypeDef * ADC_CommonInitStruct )
```

Initializes the ADCs peripherals according to the specified parameters in the ADC_CommonInitStruct.

Parameters

<i>ADC_CommonInitStruct</i>	pointer to an ADC_CommonInitTypeDef structure that contains the configuration information for All ADCs peripherals.
-----------------------------	---

Return values

<i>None</i>	
-------------	--

ADC_CommonStructInit()

```
void ADC_CommonStructInit (
    ADC_CommonInitTypeDef * ADC_CommonInitStruct )
```

Fills each ADC_CommonInitStruct member with its default value.

Parameters

<i>ADC_CommonInitStruct</i>	pointer to an ADC_CommonInitTypeDef structure which will be initialized.
-----------------------------	--

Return values

<i>None</i>	
-------------	--

ADC_DeInit()

```
void ADC_DeInit (
    void )
```

Deinitializes all ADCs peripherals registers to their default reset values.

Parameters

<i>None</i>	
-------------	--

Return values

<i>None</i>	
-------------	--

ADC_Init()

```
void ADC_Init (
    ADC_TypeDef * ADCx,
    ADC_InitTypeDef * ADC_InitStruct )
```

Initializes the ADCx peripheral according to the specified parameters in the ADC_InitStruct.

Note

This function is used to configure the global features of the ADC (Resolution and Data Alignment), however, the rest of the configuration parameters are specific to the regular channels group (scan mode activation, continuous mode activation, External trigger source and edge, number of conversion in the regular channels group sequencer).

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_InitStruct</i>	pointer to an ADC_InitTypeDef structure that contains the configuration information for the specified ADC peripheral.

Return values

<i>None</i>	
-------------	--

ADC_StructInit()

```
void ADC_StructInit (
    ADC_InitTypeDef * ADC_InitStruct )
```

Fills each ADC_InitStruct member with its default value.

Note

This function is used to initialize the global features of the ADC (Resolution and Data Alignment), however, the rest of the configuration parameters are specific to the regular channels group (scan mode activation, continuous mode activation, External trigger source and edge, number of conversion in the regular channels group sequencer).

Parameters

<code>ADC_InitStruct</code>	pointer to an ADC_InitTypeDef structure which will be initialized.
-----------------------------	--

Return values

<code>None</code>

4.1.2.4.3 Analog Watchdog configuration functions

Analog Watchdog configuration functions.

Functions

- void [ADC_AnalogWatchdogCmd](#) (ADC_TypeDef *ADCx, uint32_t ADC_AnalogWatchdog)
Enables or disables the analog watchdog on single/all regular or injected channels.
- void [ADC_AnalogWatchdogThresholdsConfig](#) (ADC_TypeDef *ADCx, uint16_t HighThreshold, uint16_t LowThreshold)
Configures the high and low thresholds of the analog watchdog.
- void [ADC_AnalogWatchdogSingleChannelConfig](#) (ADC_TypeDef *ADCx, uint8_t ADC_Channel)
Configures the analog watchdog guarded single channel.

4.1.2.4.3.1 Detailed Description

Analog Watchdog configuration functions.

```
=====
                        Analog Watchdog configuration functions
=====
```

This section provides functions allowing to configure the Analog Watchdog (AWD) feature in the ADC.

A typical configuration Analog Watchdog is done following these steps :

1. the ADC guarded channel(s) is (are) selected using the `ADC_AnalogWatchdogSingleChannelConfig()` function.
2. The Analog watchdog lower and higher threshold are configured using the `ADC_AnalogWatchdogThresholdsConfig()` function.
3. The Analog watchdog is enabled and configured to enable the check, on one or more channels, using the `ADC_AnalogWatchdogCmd()` function.

4.1.2.4.3.2 Function Documentation

ADC_AnalogWatchdogCmd()

```
void ADC_AnalogWatchdogCmd (
    ADC_TypeDef * ADCx,
    uint32_t ADC_AnalogWatchdog )
```

Enables or disables the analog watchdog on single/all regular or injected channels.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_AnalogWatchdog</i>	<p>the ADC analog watchdog configuration. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> ADC_AnalogWatchdog_SingleRegEnable: Analog watchdog on a single regular channel ADC_AnalogWatchdog_SingleInjecEnable: Analog watchdog on a single injected channel ADC_AnalogWatchdog_SingleRegOrInjecEnable: Analog watchdog on a single regular or injected channel ADC_AnalogWatchdog_AllRegEnable: Analog watchdog on all regular channel ADC_AnalogWatchdog_AllInjecEnable: Analog watchdog on all injected channel ADC_AnalogWatchdog_AllRegAllInjecEnable: Analog watchdog on all regular and injected channels ADC_AnalogWatchdog_None: No channel guarded by the analog watchdog

Return values

<i>None</i>	
-------------	--

ADC_AnalogWatchdogSingleChannelConfig()

```
void ADC_AnalogWatchdogSingleChannelConfig (
    ADC_TypeDef * ADCx,
    uint8_t ADC_Channel )
```

Configures the analog watchdog guarded single channel.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Parameters

<i>ADC_Channel</i>	<p>the ADC channel to configure for the analog watchdog. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <i>ADC_Channel_0</i>: ADC Channel0 selected • <i>ADC_Channel_1</i>: ADC Channel1 selected • <i>ADC_Channel_2</i>: ADC Channel2 selected • <i>ADC_Channel_3</i>: ADC Channel3 selected • <i>ADC_Channel_4</i>: ADC Channel4 selected • <i>ADC_Channel_5</i>: ADC Channel5 selected • <i>ADC_Channel_6</i>: ADC Channel6 selected • <i>ADC_Channel_7</i>: ADC Channel7 selected • <i>ADC_Channel_8</i>: ADC Channel8 selected • <i>ADC_Channel_9</i>: ADC Channel9 selected • <i>ADC_Channel_10</i>: ADC Channel10 selected • <i>ADC_Channel_11</i>: ADC Channel11 selected • <i>ADC_Channel_12</i>: ADC Channel12 selected • <i>ADC_Channel_13</i>: ADC Channel13 selected • <i>ADC_Channel_14</i>: ADC Channel14 selected • <i>ADC_Channel_15</i>: ADC Channel15 selected • <i>ADC_Channel_16</i>: ADC Channel16 selected • <i>ADC_Channel_17</i>: ADC Channel17 selected • <i>ADC_Channel_18</i>: ADC Channel18 selected
--------------------	---

Return values

<i>None</i>	
-------------	--

ADC_AnalogWatchdogThresholdsConfig()

```
void ADC_AnalogWatchdogThresholdsConfig (
    ADC_TypeDef * ADCx,
    uint16_t HighThreshold,
    uint16_t LowThreshold )
```

Configures the high and low thresholds of the analog watchdog.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>HighThreshold</i>	the ADC analog watchdog High threshold value. This parameter must be a 12-bit value.
<i>LowThreshold</i>	the ADC analog watchdog Low threshold value. This parameter must be a 12-bit value.

Return values

None	
------	--

4.1.2.4.4 Temperature Sensor, Vrefint (Voltage Reference internal)

Temperature Sensor, Vrefint and VBAT management functions.

Functions

- void [ADC_TempSensorVrefintCmd](#) (FunctionalState NewState)
Enables or disables the temperature sensor and Vrefint channels.
- void [ADC_VBATCmd](#) (FunctionalState NewState)
Enables or disables the VBAT (Voltage Battery) channel.

4.1.2.4.4.1 Detailed Description

Temperature Sensor, Vrefint and VBAT management functions.

and VBAT (Voltage BATtery) management functions

```
=====
                        Temperature Sensor, Vrefint and VBAT management functions
=====
```

This section provides functions allowing to enable/ disable the internal connections between the ADC and the Temperature Sensor, the Vrefint and the Vbat sources.

A typical configuration to get the Temperature sensor and Vrefint channels voltages is done following these steps :

1. Enable the internal connection of Temperature sensor and Vrefint sources with the ADC channels using `ADC_TempSensorVrefintCmd()` function.
2. Select the `ADC_Channel_TempSensor` and/or `ADC_Channel_Vrefint` using `ADC-RegularChannelConfig()` or `ADC-InjectedChannelConfig()` functions
3. Get the voltage values, using `ADC_GetConversionValue()` or `ADC_GetInjectedConversionValue()`.

A typical configuration to get the VBAT channel voltage is done following these steps :

1. Enable the internal connection of VBAT source with the ADC channel using `ADC_VBATCmd()` function.
2. Select the `ADC_Channel_Vbat` using `ADC-RegularChannelConfig()` or `ADC-InjectedChannelConfig()` functions
3. Get the voltage value, using `ADC_GetConversionValue()` or `ADC_GetInjectedConversionValue()`.

4.1.2.4.4.2 Function Documentation

ADC_TempSensorVrefintCmd()

```
void ADC_TempSensorVrefintCmd (
    FunctionalState NewState )
```

Enables or disables the temperature sensor and Vrefint channels.

Parameters

<i>NewState</i>	new state of the temperature sensor and Vrefint channels. This parameter can be: ENABLE or DISABLE.
-----------------	---

Return values

<i>None</i>	
-------------	--

ADC_VBATCmd()

```
void ADC_VBATCmd (
    FunctionalState NewState )
```

Enables or disables the VBAT (Voltage Battery) channel.

Parameters

<i>NewState</i>	new state of the VBAT channel. This parameter can be: ENABLE or DISABLE.
-----------------	--

Return values

<i>None</i>	
-------------	--

4.1.2.4.5 Regular Channels Configuration functions

Regular Channels Configuration functions.

Functions

- void [ADC_RegularChannelConfig](#) (ADC_TypeDef *ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)
Configures for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
- void [ADC_SoftwareStartConv](#) (ADC_TypeDef *ADCx)
Enables the selected ADC software start conversion of the regular channels.
- FlagStatus [ADC_GetSoftwareStartConvStatus](#) (ADC_TypeDef *ADCx)
Gets the selected ADC Software start regular conversion Status.
- void [ADC_EOCOnEachRegularChannelCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the EOC on each regular channel conversion.
- void [ADC_ContinuousModeCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the ADC continuous conversion mode.
- void [ADC_DiscModeChannelCountConfig](#) (ADC_TypeDef *ADCx, uint8_t Number)
Configures the discontinuous mode for the selected ADC regular group channel.
- void [ADC_DiscModeCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the discontinuous mode on regular group channel for the specified ADC.
- uint16_t [ADC_GetConversionValue](#) (ADC_TypeDef *ADCx)
Returns the last ADCx conversion result data for regular channel.
- uint32_t [ADC_GetMultiModeConversionValue](#) (void)
Returns the last ADC1, ADC2 and ADC3 regular conversions results data in the selected multi mode.

4.1.2.4.5.1 Detailed Description

Regular Channels Configuration functions.

```
=====
                        Regular Channels Configuration functions
=====
```

This section provides functions allowing to manage the ADC's regular channels, it is composed of 2 sub sections :

1. Configuration and management functions for regular channels: This subsection provides functions allowing to configure the ADC regular channels :
 - Configure the rank in the regular group sequencer for each channel
 - Configure the sampling time for each channel
 - select the conversion Trigger for regular channels
 - select the desired EOC event behavior configuration
 - Activate the continuous Mode (*)
 - Activate the Discontinuous Mode

Please Note that the following features for regular channels are configured using the ADC_Init() function :

 - scan mode activation
 - continuous mode activation (**)
 - External trigger source
 - External trigger edge
 - number of conversion in the regular channels group sequencer.

@note (*) and (**) are performing the same configuration
2. Get the conversion data: This subsection provides an important function in the ADC peripheral since it returns the converted data of the current regular channel. When the Conversion value is read, the EOC Flag is automatically cleared.

@note For multi ADC mode, the last ADC1, ADC2 and ADC3 regular conversions results data (in the selected multi mode) can be returned in the same time using ADC_GetMultiModeConversionValue() function.

4.1.2.4.5.2 Function Documentation

ADC_ContinuousModeCmd()

```
void ADC_ContinuousModeCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the ADC continuous conversion mode.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC continuous conversion mode This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

ADC_DiscModeChannelCountConfig()

```
void ADC_DiscModeChannelCountConfig (
    ADC_TypeDef * ADCx,
    uint8_t Number )
```

Configures the discontinuous mode for the selected ADC regular group channel.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>Number</i>	specifies the discontinuous mode regular channel count value. This number must be between 1 and 8.

Return values

<i>None</i>	
-------------	--

ADC_DiscModeCmd()

```
void ADC_DiscModeCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the discontinuous mode on regular group channel for the specified ADC.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC discontinuous mode on regular group channel. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

ADC_EOCOnEachRegularChannelCmd()

```
void ADC_EOCOnEachRegularChannelCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the EOC on each regular channel conversion.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC EOC flag rising This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

ADC_GetConversionValue()

```
uint16_t ADC_GetConversionValue (
    ADC_TypeDef * ADCx )
```

Returns the last ADCx conversion result data for regular channel.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Return values

<i>The</i>	Data conversion value.
------------	------------------------

ADC_GetMultiModeConversionValue()

```
uint32_t ADC_GetMultiModeConversionValue (
    void )
```

Returns the last ADC1, ADC2 and ADC3 regular conversions results data in the selected multi mode.

Parameters

<i>None</i>	
-------------	--

Return values

<i>The</i>	Data conversion value.
------------	------------------------

Note

In dual mode, the value returned by this function is as following Data[15:0] : these bits contain the regular data of ADC1. Data[31:16]: these bits contain the regular data of ADC2.

In triple mode, the value returned by this function is as following Data[15:0] : these bits contain alternatively the regular data of ADC1, ADC3 and ADC2. Data[31:16]: these bits contain alternatively the regular data of ADC2, ADC1 and ADC3.

ADC_GetSoftwareStartConvStatus()

```
FlagStatus ADC_GetSoftwareStartConvStatus (
    ADC_TypeDef * ADCx )
```

Gets the selected ADC Software start regular conversion Status.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Return values

<i>The</i>	new state of ADC software start conversion (SET or RESET).
------------	--

ADC_RegularChannelConfig()

```
void ADC_RegularChannelConfig (
    ADC_TypeDef * ADCx,
    uint8_t ADC_Channel,
    uint8_t Rank,
    uint8_t ADC_SampleTime )
```

Configures for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Parameters

<i>ADC_Channel</i>	<p>the ADC channel to configure. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • ADC_Channel_0: ADC Channel0 selected • ADC_Channel_1: ADC Channel1 selected • ADC_Channel_2: ADC Channel2 selected • ADC_Channel_3: ADC Channel3 selected • ADC_Channel_4: ADC Channel4 selected • ADC_Channel_5: ADC Channel5 selected • ADC_Channel_6: ADC Channel6 selected • ADC_Channel_7: ADC Channel7 selected • ADC_Channel_8: ADC Channel8 selected • ADC_Channel_9: ADC Channel9 selected • ADC_Channel_10: ADC Channel10 selected • ADC_Channel_11: ADC Channel11 selected • ADC_Channel_12: ADC Channel12 selected • ADC_Channel_13: ADC Channel13 selected • ADC_Channel_14: ADC Channel14 selected • ADC_Channel_15: ADC Channel15 selected • ADC_Channel_16: ADC Channel16 selected • ADC_Channel_17: ADC Channel17 selected • ADC_Channel_18: ADC Channel18 selected
<i>Rank</i>	The rank in the regular group sequencer. This parameter must be between 1 to 16.
<i>ADC_SampleTime</i>	<p>The sample time value to be set for the selected channel. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • ADC_SampleTime_3Cycles: Sample time equal to 3 cycles • ADC_SampleTime_15Cycles: Sample time equal to 15 cycles • ADC_SampleTime_28Cycles: Sample time equal to 28 cycles • ADC_SampleTime_56Cycles: Sample time equal to 56 cycles • ADC_SampleTime_84Cycles: Sample time equal to 84 cycles • ADC_SampleTime_112Cycles: Sample time equal to 112 cycles • ADC_SampleTime_144Cycles: Sample time equal to 144 cycles • ADC_SampleTime_480Cycles: Sample time equal to 480 cycles

Return values

None	
------	--

ADC_SoftwareStartConv()

```
void ADC_SoftwareStartConv (
    ADC_TypeDef * ADCx )
```

Enables the selected ADC software start conversion of the regular channels.

Parameters

ADCx	where x can be 1, 2 or 3 to select the ADC peripheral.
------	--

Return values

None	
------	--

4.1.2.4.6 Regular Channels DMA Configuration functions

Regular Channels DMA Configuration functions.

Functions

- void [ADC_DMACmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the specified ADC DMA request.
- void [ADC_DMAREquestAfterLastTransferCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the ADC DMA request after last transfer (Single-ADC mode)
- void [ADC_MultiModeDMAREquestAfterLastTransferCmd](#) (FunctionalState NewState)
Enables or disables the ADC DMA request after last transfer in multi ADC mode

4.1.2.4.6.1 Detailed Description

Regular Channels DMA Configuration functions.

```
=====
Regular Channels DMA Configuration functions
=====
```

This section provides functions allowing to configure the DMA for ADC regular channels.
Since converted regular channel values are stored into a unique data register, it is useful to use DMA for conversion of more than one regular channel. This avoids the loss of the data already stored in the ADC Data register.

When the DMA mode is enabled (using the `ADC_DMACmd()` function), after each

conversion of a regular channel, a DMA request is generated.

Depending on the "DMA disable selection for Independent ADC mode" configuration (using the `ADC_DMARequestAfterLastTransferCmd()` function), at the end of the last DMA transfer, two possibilities are allowed:

- No new DMA request is issued to the DMA controller (feature DISABLED)
- Requests can continue to be generated (feature ENABLED).

Depending on the "DMA disable selection for multi ADC mode" configuration (using the void `ADC_MultiModeDMARequestAfterLastTransferCmd()` function), at the end of the last DMA transfer, two possibilities are allowed:

- No new DMA request is issued to the DMA controller (feature DISABLED)
- Requests can continue to be generated (feature ENABLED).

4.1.2.4.6.2 Function Documentation

ADC_DMAMCmd()

```
void ADC_DMAMCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the specified ADC DMA request.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC DMA transfer. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

ADC_DMARequestAfterLastTransferCmd()

```
void ADC_DMARequestAfterLastTransferCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the ADC DMA request after last transfer (Single-ADC mode)

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC DMA request after last transfer. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

ADC_MultiModeDMARequestAfterLastTransferCmd()

```
void ADC_MultiModeDMARequestAfterLastTransferCmd (
    FunctionalState NewState )
```

Enables or disables the ADC DMA request after last transfer in multi ADC mode

Parameters

NewState	new state of the selected ADC DMA request after last transfer. This parameter can be: ENABLE or DISABLE.
-----------------	--

Note

if Enabled, DMA requests are issued as long as data are converted and DMA mode for multi ADC mode (selected using [ADC_CommonInit\(\)](#) function by ADC_CommonInitStruct.ADC_DMAAccessMode structure member) is ADC_DMAAccessMode_1, ADC_DMAAccessMode_2 or ADC_DMAAccessMode_3.

Return values

None	
-------------	--

4.1.2.4.7 Injected channels Configuration functions

Injected channels Configuration functions.

Functions

- void [ADC_InjectedChannelConfig](#) (ADC_TypeDef *ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)
Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time.
- void [ADC_InjectedSequencerLengthConfig](#) (ADC_TypeDef *ADCx, uint8_t Length)
Configures the sequencer length for injected channels.
- void [ADC_SetInjectedOffset](#) (ADC_TypeDef *ADCx, uint8_t ADC_InjectedChannel, uint16_t Offset)
Set the injected channels conversion value offset.
- void [ADC_ExternalTrigInjectedConvConfig](#) (ADC_TypeDef *ADCx, uint32_t ADC_ExternalTrigInjecConv)
Configures the ADCx external trigger for injected channels conversion.
- void [ADC_ExternalTrigInjectedConvEdgeConfig](#) (ADC_TypeDef *ADCx, uint32_t ADC_ExternalTrigInjecConvEdge)
Configures the ADCx external trigger edge for injected channels conversion.
- void [ADC_SoftwareStartInjectedConv](#) (ADC_TypeDef *ADCx)
Enables the selected ADC software start conversion of the injected channels.
- FlagStatus [ADC_GetSoftwareStartInjectedConvCmdStatus](#) (ADC_TypeDef *ADCx)
Gets the selected ADC Software start injected conversion Status.
- void [ADC_AutoInjectedConvCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the selected ADC automatic injected group conversion after regular one.
- void [ADC_InjectedDiscModeCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the discontinuous mode for injected group channel for the specified ADC.
- uint16_t [ADC_GetInjectedConversionValue](#) (ADC_TypeDef *ADCx, uint8_t ADC_InjectedChannel)
Returns the ADC injected channel conversion result.

4.1.2.4.7.1 Detailed Description

Injected channels Configuration functions.

```
=====
                        Injected channels Configuration functions
=====
```

This section provide functions allowing to configure the ADC Injected channels, it is composed of 2 sub sections :

1. Configuration functions for Injected channels: This subsection provides functions allowing to configure the ADC injected channels :
 - Configure the rank in the injected group sequencer for each channel
 - Configure the sampling time for each channel
 - Activate the Auto injected Mode
 - Activate the Discontinuous Mode
 - scan mode activation
 - External/software trigger source
 - External trigger edge
 - injected channels sequencer.
2. Get the Specified Injected channel conversion data: This subsection provides an important function in the ADC peripheral since it returns the converted data of the specific injected channel.

4.1.2.4.7.2 Function Documentation

ADC_AutoInjectedConvCmd()

```
void ADC_AutoInjectedConvCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the selected ADC automatic injected group conversion after regular one.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC auto injected conversion This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

ADC_ExternalTrigInjectedConvConfig()

```
void ADC_ExternalTrigInjectedConvConfig (
    ADC_TypeDef * ADCx,
    uint32_t ADC_ExternalTrigInjecConv )
```

Configures the ADCx external trigger for injected channels conversion.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_ExternalTrigInjecConv</i>	<p>specifies the ADC trigger to start injected conversion. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <i>ADC_ExternalTrigInjecConv_T1_CC4</i>: Timer1 capture compare4 selected • <i>ADC_ExternalTrigInjecConv_T1_TRGO</i>: Timer1 TRGO event selected • <i>ADC_ExternalTrigInjecConv_T2_CC1</i>: Timer2 capture compare1 selected • <i>ADC_ExternalTrigInjecConv_T2_TRGO</i>: Timer2 TRGO event selected • <i>ADC_ExternalTrigInjecConv_T3_CC2</i>: Timer3 capture compare2 selected • <i>ADC_ExternalTrigInjecConv_T3_CC4</i>: Timer3 capture compare4 selected • <i>ADC_ExternalTrigInjecConv_T4_CC1</i>: Timer4 capture compare1 selected • <i>ADC_ExternalTrigInjecConv_T4_CC2</i>: Timer4 capture compare2 selected • <i>ADC_ExternalTrigInjecConv_T4_CC3</i>: Timer4 capture compare3 selected • <i>ADC_ExternalTrigInjecConv_T4_TRGO</i>: Timer4 TRGO event selected • <i>ADC_ExternalTrigInjecConv_T5_CC4</i>: Timer5 capture compare4 selected • <i>ADC_ExternalTrigInjecConv_T5_TRGO</i>: Timer5 TRGO event selected • <i>ADC_ExternalTrigInjecConv_T8_CC2</i>: Timer8 capture compare2 selected • <i>ADC_ExternalTrigInjecConv_T8_CC3</i>: Timer8 capture compare3 selected • <i>ADC_ExternalTrigInjecConv_T8_CC4</i>: Timer8 capture compare4 selected • <i>ADC_ExternalTrigInjecConv_Ext_IT15</i>: External interrupt line 15 event selected

Return values

<i>None</i>	
-------------	--

ADC_ExternalTrigInjectedConvEdgeConfig()

```
void ADC_ExternalTrigInjectedConvEdgeConfig (
    ADC_TypeDef * ADCx,
    uint32_t ADC_ExternalTrigInjecConvEdge )
```

Configures the ADCx external trigger edge for injected channels conversion.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_ExternalTrigInjecConvEdge</i>	<p>specifies the ADC external trigger edge to start injected conversion. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • ADC_ExternalTrigInjecConvEdge_None: external trigger disabled for injected conversion • ADC_ExternalTrigInjecConvEdge_Rising: detection on rising edge • ADC_ExternalTrigInjecConvEdge_Falling: detection on falling edge • ADC_ExternalTrigInjecConvEdge_RisingFalling: detection on both rising and falling edge

Return values

<i>None</i>	
-------------	--

ADC_GetInjectedConversionValue()

```
uint16_t ADC_GetInjectedConversionValue (
    ADC_TypeDef * ADCx,
    uint8_t ADC_InjectedChannel )
```

Returns the ADC injected channel conversion result.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_InjectedChannel</i>	<p>the converted ADC injected channel. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • ADC_InjectedChannel_1: Injected Channel1 selected • ADC_InjectedChannel_2: Injected Channel2 selected • ADC_InjectedChannel_3: Injected Channel3 selected • ADC_InjectedChannel_4: Injected Channel4 selected

Return values

<i>The</i>	Data conversion value.
------------	------------------------

ADC_GetSoftwareStartInjectedConvCmdStatus()

```
FlagStatus ADC_GetSoftwareStartInjectedConvCmdStatus (
    ADC_TypeDef * ADCx )
```

Gets the selected ADC Software start injected conversion Status.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Return values

<i>The</i>	new state of ADC software start injected conversion (SET or RESET).
------------	---

ADC_InjectedChannelConfig()

```
void ADC_InjectedChannelConfig (
    ADC_TypeDef * ADCx,
    uint8_t ADC_Channel,
    uint8_t Rank,
    uint8_t ADC_SampleTime )
```

Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Parameters

<i>ADC_Channel</i>	<p>the ADC channel to configure. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>ADC_Channel_0</code>: ADC Channel0 selected • <code>ADC_Channel_1</code>: ADC Channel1 selected • <code>ADC_Channel_2</code>: ADC Channel2 selected • <code>ADC_Channel_3</code>: ADC Channel3 selected • <code>ADC_Channel_4</code>: ADC Channel4 selected • <code>ADC_Channel_5</code>: ADC Channel5 selected • <code>ADC_Channel_6</code>: ADC Channel6 selected • <code>ADC_Channel_7</code>: ADC Channel7 selected • <code>ADC_Channel_8</code>: ADC Channel8 selected • <code>ADC_Channel_9</code>: ADC Channel9 selected • <code>ADC_Channel_10</code>: ADC Channel10 selected • <code>ADC_Channel_11</code>: ADC Channel11 selected • <code>ADC_Channel_12</code>: ADC Channel12 selected • <code>ADC_Channel_13</code>: ADC Channel13 selected • <code>ADC_Channel_14</code>: ADC Channel14 selected • <code>ADC_Channel_15</code>: ADC Channel15 selected • <code>ADC_Channel_16</code>: ADC Channel16 selected • <code>ADC_Channel_17</code>: ADC Channel17 selected • <code>ADC_Channel_18</code>: ADC Channel18 selected
<i>Rank</i>	The rank in the injected group sequencer. This parameter must be between 1 to 4.
<i>ADC_SampleTime</i>	<p>The sample time value to be set for the selected channel. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>ADC_SampleTime_3Cycles</code>: Sample time equal to 3 cycles • <code>ADC_SampleTime_15Cycles</code>: Sample time equal to 15 cycles • <code>ADC_SampleTime_28Cycles</code>: Sample time equal to 28 cycles • <code>ADC_SampleTime_56Cycles</code>: Sample time equal to 56 cycles • <code>ADC_SampleTime_84Cycles</code>: Sample time equal to 84 cycles • <code>ADC_SampleTime_112Cycles</code>: Sample time equal to 112 cycles • <code>ADC_SampleTime_144Cycles</code>: Sample time equal to 144 cycles • <code>ADC_SampleTime_480Cycles</code>: Sample time equal to 480 cycles

Return values

<i>None</i>	
-------------	--

ADC_InjectedDiscModeCmd()

```
void ADC_InjectedDiscModeCmd (
    ADC_TypeDef * ADCx,
    FunctionalState NewState )
```

Enables or disables the discontinuous mode for injected group channel for the specified ADC.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>NewState</i>	new state of the selected ADC discontinuous mode on injected group channel. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

ADC_InjectedSequencerLengthConfig()

```
void ADC_InjectedSequencerLengthConfig (
    ADC_TypeDef * ADCx,
    uint8_t Length )
```

Configures the sequencer length for injected channels.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>Length</i>	The sequencer length. This parameter must be a number between 1 to 4.

Return values

<i>None</i>	
-------------	--

ADC_SetInjectedOffset()

```
void ADC_SetInjectedOffset (
    ADC_TypeDef * ADCx,
    uint8_t ADC_InjectedChannel,
    uint16_t Offset )
```

Set the injected channels conversion value offset.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_InjectedChannel</i>	the ADC injected channel to set its offset. This parameter can be one of the following values: <ul style="list-style-type: none"> • ADC_InjectedChannel_1: Injected Channel1 selected • ADC_InjectedChannel_2: Injected Channel2 selected • ADC_InjectedChannel_3: Injected Channel3 selected • ADC_InjectedChannel_4: Injected Channel4 selected
<i>Offset</i>	the offset value for the selected ADC injected channel This parameter must be a 12bit value.

Return values

<i>None</i>	
-------------	--

ADC_SoftwareStartInjectedConv()

```
void ADC_SoftwareStartInjectedConv (
    ADC_TypeDef * ADCx )
```

Enables the selected ADC software start conversion of the injected channels.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Return values

<i>None</i>	
-------------	--

4.1.2.4.8 Interrupts and flags management functions

Interrupts and flags management functions.

Functions

- void [ADC_ITConfig](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT, FunctionalState NewState)
Enables or disables the specified ADC interrupts.
- FlagStatus [ADC_GetFlagStatus](#) (ADC_TypeDef *ADCx, uint8_t ADC_FLAG)
Checks whether the specified ADC flag is set or not.
- void [ADC_ClearFlag](#) (ADC_TypeDef *ADCx, uint8_t ADC_FLAG)
Clears the ADCx's pending flags.
- ITStatus [ADC_GetITStatus](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT)
Checks whether the specified ADC interrupt has occurred or not.
- void [ADC_ClearITPendingBit](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT)
Clears the ADCx's interrupt pending bits.

4.1.2.4.8.1 Detailed Description

Interrupts and flags management functions.

```

=====
                        Interrupts and flags management functions
=====

This section provides functions allowing to configure the ADC Interrupts and
to get the status and clear flags and Interrupts pending bits.

Each ADC provides 4 Interrupts sources and 6 Flags which can be divided into
3 groups:

I. Flags and Interrupts for ADC regular channels
=====
Flags :
-----
    1. ADC_FLAG_OVR : Overrun detection when regular converted data are lost

    2. ADC_FLAG_EOC : Regular channel end of conversion ==> to indicate (depending
        on EOCS bit, managed by ADC_EOOnEachRegularChannelCmd() ) the end of:
        ==> a regular CHANNEL conversion
        ==> sequence of regular GROUP conversions .

    3. ADC_FLAG_STRT: Regular channel start ==> to indicate when regular CHANNEL
        conversion starts.

Interrupts :
-----
    1. ADC_IT_OVR : specifies the interrupt source for Overrun detection event.
    2. ADC_IT_EOC : specifies the interrupt source for Regular channel end of
        conversion event.

II. Flags and Interrupts for ADC Injected channels
=====
Flags :
-----
    1. ADC_FLAG_JEOC : Injected channel end of conversion ==> to indicate at
        the end of injected GROUP conversion

    2. ADC_FLAG_JSTRT: Injected channel start ==> to indicate hardware when
        injected GROUP conversion starts.

Interrupts :
-----
    1. ADC_IT_JEOC : specifies the interrupt source for Injected channel end of
        conversion event.

III. General Flags and Interrupts for the ADC
=====
Flags :
-----
    1. ADC_FLAG_AWD: Analog watchdog ==> to indicate if the converted voltage
        crosses the programmed thresholds values.

Interrupts :
-----
    1. ADC_IT_AWD : specifies the interrupt source for Analog watchdog event.

The user should identify which mode will be used in his application to manage
the ADC controller events: Polling mode or Interrupt mode.

In the Polling Mode it is advised to use the following functions:
    - ADC_GetFlagStatus() : to check if flags events occur.
    - ADC_ClearFlag()     : to clear the flags events.

In the Interrupt Mode it is advised to use the following functions:
    - ADC_ITConfig()      : to enable or disable the interrupt source.

```

- ADC_GetITStatus() : to check if Interrupt occurs.
- ADC_ClearITPendingBit() : to clear the Interrupt pending Bit (corresponding Flag).

4.1.2.4.8.2 Function Documentation

ADC_ClearFlag()

```
void ADC_ClearFlag (
    ADC_TypeDef * ADCx,
    uint8_t ADC_FLAG )
```

Clears the ADCx's pending flags.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_FLAG</i>	specifies the flag to clear. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • ADC_FLAG_AWD: Analog watchdog flag • ADC_FLAG_EOC: End of conversion flag • ADC_FLAG_JEOC: End of injected group conversion flag • ADC_FLAG_JSTRT: Start of injected group conversion flag • ADC_FLAG_STRT: Start of regular group conversion flag • ADC_FLAG_OVR: Overrun flag

Return values

<i>None</i>	
-------------	--

ADC_ClearITPendingBit()

```
void ADC_ClearITPendingBit (
    ADC_TypeDef * ADCx,
    uint16_t ADC_IT )
```

Clears the ADCx's interrupt pending bits.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
-------------	--

Parameters

<i>ADC</i> ↔ <i>_IT</i>	specifies the ADC interrupt pending bit to clear. This parameter can be one of the following values: <ul style="list-style-type: none"> • ADC_IT_EOC: End of conversion interrupt mask • ADC_IT_AWD: Analog watchdog interrupt mask • ADC_IT_JEOC: End of injected conversion interrupt mask • ADC_IT_OVR: Overrun interrupt mask
----------------------------	---

Return values

<i>None</i>	
-------------	--

ADC_GetFlagStatus()

```
FlagStatus ADC_GetFlagStatus (
    ADC_TypeDef * ADCx,
    uint8_t ADC_FLAG )
```

Checks whether the specified ADC flag is set or not.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_FLAG</i>	specifies the flag to check. This parameter can be one of the following values: <ul style="list-style-type: none"> • ADC_FLAG_AWD: Analog watchdog flag • ADC_FLAG_EOC: End of conversion flag • ADC_FLAG_JEOC: End of injected group conversion flag • ADC_FLAG_JSTRT: Start of injected group conversion flag • ADC_FLAG_STRT: Start of regular group conversion flag • ADC_FLAG_OVR: Overrun flag

Return values

<i>The</i>	new state of ADC_FLAG (SET or RESET).
------------	---------------------------------------

ADC_GetITStatus()

```
ITStatus ADC_GetITStatus (
    ADC_TypeDef * ADCx,
    uint16_t ADC_IT )
```

Checks whether the specified ADC interrupt has occurred or not.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_IT</i>	specifies the ADC interrupt source to check. This parameter can be one of the following values: <ul style="list-style-type: none"> • ADC_IT_EOC: End of conversion interrupt mask • ADC_IT_AWD: Analog watchdog interrupt mask • ADC_IT_JEOC: End of injected conversion interrupt mask • ADC_IT_OVR: Overrun interrupt mask

Return values

<i>The</i>	new state of ADC_IT (SET or RESET).
------------	-------------------------------------

ADC_ITConfig()

```
void ADC_ITConfig (
    ADC_TypeDef * ADCx,
    uint16_t ADC_IT,
    FunctionalState NewState )
```

Enables or disables the specified ADC interrupts.

Parameters

<i>ADCx</i>	where x can be 1, 2 or 3 to select the ADC peripheral.
<i>ADC_IT</i>	specifies the ADC interrupt sources to be enabled or disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> • ADC_IT_EOC: End of conversion interrupt mask • ADC_IT_AWD: Analog watchdog interrupt mask • ADC_IT_JEOC: End of injected conversion interrupt mask • ADC_IT_OVR: Overrun interrupt enable
<i>NewState</i>	new state of the specified ADC interrupts. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.2.5 ADC_Exported_Constants

Modules

- [ADC_Common_mode](#)
- [ADC_Prescaler](#)
- [ADC_Direct_memory_access_mode_for_multi_mode](#)
- [ADC_delay_between_2_sampling_phases](#)
- [ADC_resolution](#)
- [ADC_external_trigger_edge_for_regular_channels_conversion](#)
- [ADC_extrenal_trigger_sources_for_regular_channels_conversion](#)
- [ADC_data_align](#)
- [ADC_channels](#)
- [ADC_sampling_times](#)
- [ADC_external_trigger_edge_for_injected_channels_conversion](#)
- [ADC_extrenal_trigger_sources_for_injected_channels_conversion](#)
- [ADC_injected_channel_selection](#)
- [ADC_analog_watchdog_selection](#)
- [ADC_interrupts_definition](#)
- [ADC_flags_definition](#)
- [ADC_thresholds](#)
- [ADC_injected_offset](#)
- [ADC_injected_length](#)
- [ADC_injected_rank](#)
- [ADC_regular_length](#)
- [ADC_regular_rank](#)
- [ADC_regular_discontinuous_mode_number](#)

Macros

- [#define IS_ADC_ALL_PERIPH\(PERIPH\)](#)

4.1.2.5.1 Detailed Description

4.1.2.5.2 Macro Definition Documentation

4.1.2.5.2.1 IS_ADC_ALL_PERIPH

```
#define IS_ADC_ALL_PERIPH(  
    PERIPH )
```

Value:

```
(( (PERIPH) == ADC1) || \  
((PERIPH) == ADC2) || \  
((PERIPH) == ADC3))
```

4.1.2.5.3 ADC_Common_mode

Macros

- #define `ADC_Mode_Independent` ((uint32_t)0x00000000)
- #define `ADC_DualMode_RegSimult_InjecSimult` ((uint32_t)0x00000001)
- #define `ADC_DualMode_RegSimult_AlterTrig` ((uint32_t)0x00000002)
- #define `ADC_DualMode_InjecSimult` ((uint32_t)0x00000005)
- #define `ADC_DualMode_RegSimult` ((uint32_t)0x00000006)
- #define `ADC_DualMode_Interl` ((uint32_t)0x00000007)
- #define `ADC_DualMode_AlterTrig` ((uint32_t)0x00000009)
- #define `ADC_TripleMode_RegSimult_InjecSimult` ((uint32_t)0x00000011)
- #define `ADC_TripleMode_RegSimult_AlterTrig` ((uint32_t)0x00000012)
- #define `ADC_TripleMode_InjecSimult` ((uint32_t)0x00000015)
- #define `ADC_TripleMode_RegSimult` ((uint32_t)0x00000016)
- #define `ADC_TripleMode_Interl` ((uint32_t)0x00000017)
- #define `ADC_TripleMode_AlterTrig` ((uint32_t)0x00000019)
- #define `IS_ADC_MODE`(MODE)

4.1.2.5.3.1 Detailed Description

4.1.2.5.3.2 Macro Definition Documentation

ADC_DualMode_AlterTrig

```
#define ADC_DualMode_AlterTrig ((uint32_t)0x00000009)
```

ADC_DualMode_InjecSimult

```
#define ADC_DualMode_InjecSimult ((uint32_t)0x00000005)
```

ADC_DualMode_Interl

```
#define ADC_DualMode_Interl ((uint32_t)0x00000007)
```

ADC_DualMode_RegSimult

```
#define ADC_DualMode_RegSimult ((uint32_t)0x00000006)
```

ADC_DualMode_RegSimult_AlterTrig

```
#define ADC_DualMode_RegSimult_AlterTrig ((uint32_t)0x00000002)
```

ADC_DualMode_RegSimult_InjecSimult

```
#define ADC_DualMode_RegSimult_InjecSimult ((uint32_t)0x00000001)
```

ADC_Mode_Independent

```
#define ADC_Mode_Independent ((uint32_t)0x00000000)
```

ADC_TripleMode_AlterTrig

```
#define ADC_TripleMode_AlterTrig ((uint32_t)0x00000019)
```

ADC_TripleMode_InjecSimult

```
#define ADC_TripleMode_InjecSimult ((uint32_t)0x00000015)
```

ADC_TripleMode_Interl

```
#define ADC_TripleMode_Interl ((uint32_t)0x00000017)
```

ADC_TripleMode_RegSimult

```
#define ADC_TripleMode_RegSimult ((uint32_t)0x00000016)
```

ADC_TripleMode_RegSimult_AlterTrig

```
#define ADC_TripleMode_RegSimult_AlterTrig ((uint32_t)0x00000012)
```

ADC_TripleMode_RegSimult_InjecSimult

```
#define ADC_TripleMode_RegSimult_InjecSimult ((uint32_t)0x00000011)
```

IS_ADC_MODE

```
#define IS_ADC_MODE(  
    MODE )
```

Value:

```
((MODE) == ADC_Mode_Independent) || \
((MODE) == ADC_DualMode_RegSimult_InjecSimult) || \
((MODE) == ADC_DualMode_RegSimult_AlterTrig) || \
((MODE) == ADC_DualMode_InjecSimult) || \
((MODE) == ADC_DualMode_RegSimult) || \
((MODE) == ADC_DualMode_Interl) || \
((MODE) == ADC_DualMode_AlterTrig) || \
((MODE) == ADC_TripleMode_RegSimult_InjecSimult) || \
((MODE) == ADC_TripleMode_RegSimult_AlterTrig) || \
((MODE) == ADC_TripleMode_InjecSimult) || \
((MODE) == ADC_TripleMode_RegSimult) || \
((MODE) == ADC_TripleMode_Interl) || \
((MODE) == ADC_TripleMode_AlterTrig))
```

4.1.2.5.4 ADC_Prescaler

Macros

- #define [ADC_Prescaler_Div2](#) ((uint32_t)0x00000000)
- #define [ADC_Prescaler_Div4](#) ((uint32_t)0x00010000)
- #define [ADC_Prescaler_Div6](#) ((uint32_t)0x00020000)
- #define [ADC_Prescaler_Div8](#) ((uint32_t)0x00030000)
- #define [IS_ADC_PRESCALER](#)(PRESCALER)

4.1.2.5.4.1 Detailed Description

4.1.2.5.4.2 Macro Definition Documentation

ADC_Prescaler_Div2

```
#define ADC_Prescaler_Div2 ((uint32_t)0x00000000)
```

ADC_Prescaler_Div4

```
#define ADC_Prescaler_Div4 ((uint32_t)0x00010000)
```

ADC_Prescaler_Div6

```
#define ADC_Prescaler_Div6 ((uint32_t)0x00020000)
```

ADC_Prescaler_Div8

```
#define ADC_Prescaler_Div8 ((uint32_t)0x00030000)
```

IS_ADC_PRESCALER

```
#define IS_ADC_PRESCALER(  
    PRESCALER )
```

Value:

```
(( (PRESCALER) == ADC\_Prescaler\_Div2) || \  
(( (PRESCALER) == ADC\_Prescaler\_Div4) || \  
(( (PRESCALER) == ADC\_Prescaler\_Div6) || \  
(( (PRESCALER) == ADC\_Prescaler\_Div8)))
```

4.1.2.5.5 ADC_Direct_memory_access_mode_for_multi_mode

Macros

- #define `ADC_DMAAccessMode_Disabled` ((uint32_t)0x00000000) /* DMA mode disabled */
- #define `ADC_DMAAccessMode_1` ((uint32_t)0x00004000) /* DMA mode 1 enabled (2 / 3 half-words one by one - 1 then 2 then 3)*/
- #define `ADC_DMAAccessMode_2` ((uint32_t)0x00008000) /* DMA mode 2 enabled (2 / 3 half-words by pairs - 2&1 then 1&3 then 3&2)*/
- #define `ADC_DMAAccessMode_3` ((uint32_t)0x0000C000) /* DMA mode 3 enabled (2 / 3 bytes by pairs - 2&1 then 1&3 then 3&2) */
- #define `IS_ADC_DMA_ACCESS_MODE`(MODE)

4.1.2.5.5.1 Detailed Description

4.1.2.5.5.2 Macro Definition Documentation

ADC_DMAAccessMode_1

```
#define ADC_DMAAccessMode_1 ((uint32_t)0x00004000) /* DMA mode 1 enabled (2 / 3 half-words one by one - 1 then 2 then 3)*/
```

ADC_DMAAccessMode_2

```
#define ADC_DMAAccessMode_2 ((uint32_t)0x00008000) /* DMA mode 2 enabled (2 / 3 half-words by pairs - 2&1 then 1&3 then 3&2)*/
```

ADC_DMAAccessMode_3

```
#define ADC_DMAAccessMode_3 ((uint32_t)0x0000C000) /* DMA mode 3 enabled (2 / 3 bytes by pairs - 2&1 then 1&3 then 3&2) */
```

ADC_DMAAccessMode_Disabled

```
#define ADC_DMAAccessMode_Disabled ((uint32_t)0x00000000) /* DMA mode disabled */
```

IS_ADC_DMA_ACCESS_MODE

```
#define IS_ADC_DMA_ACCESS_MODE(  
    MODE )
```

Value:

```
((MODE) == ADC_DMAAccessMode_Disabled) || \  
((MODE) == ADC_DMAAccessMode_1) || \  
((MODE) == ADC_DMAAccessMode_2) || \  
((MODE) == ADC_DMAAccessMode_3)
```

4.1.2.5.6 ADC_delay_between_2_sampling_phases

Macros

- `#define ADC_TwoSamplingDelay_5Cycles ((uint32_t)0x00000000)`
- `#define ADC_TwoSamplingDelay_6Cycles ((uint32_t)0x00000100)`
- `#define ADC_TwoSamplingDelay_7Cycles ((uint32_t)0x00000200)`
- `#define ADC_TwoSamplingDelay_8Cycles ((uint32_t)0x00000300)`
- `#define ADC_TwoSamplingDelay_9Cycles ((uint32_t)0x00000400)`
- `#define ADC_TwoSamplingDelay_10Cycles ((uint32_t)0x00000500)`
- `#define ADC_TwoSamplingDelay_11Cycles ((uint32_t)0x00000600)`
- `#define ADC_TwoSamplingDelay_12Cycles ((uint32_t)0x00000700)`
- `#define ADC_TwoSamplingDelay_13Cycles ((uint32_t)0x00000800)`
- `#define ADC_TwoSamplingDelay_14Cycles ((uint32_t)0x00000900)`
- `#define ADC_TwoSamplingDelay_15Cycles ((uint32_t)0x00000A00)`
- `#define ADC_TwoSamplingDelay_16Cycles ((uint32_t)0x00000B00)`
- `#define ADC_TwoSamplingDelay_17Cycles ((uint32_t)0x00000C00)`
- `#define ADC_TwoSamplingDelay_18Cycles ((uint32_t)0x00000D00)`
- `#define ADC_TwoSamplingDelay_19Cycles ((uint32_t)0x00000E00)`
- `#define ADC_TwoSamplingDelay_20Cycles ((uint32_t)0x00000F00)`
- `#define IS_ADC_SAMPLING_DELAY(Delay)`

4.1.2.5.6.1 Detailed Description

4.1.2.5.6.2 Macro Definition Documentation

ADC_TwoSamplingDelay_10Cycles

```
#define ADC_TwoSamplingDelay_10Cycles ((uint32_t)0x00000500)
```

ADC_TwoSamplingDelay_11Cycles

```
#define ADC_TwoSamplingDelay_11Cycles ((uint32_t)0x00000600)
```

ADC_TwoSamplingDelay_12Cycles

```
#define ADC_TwoSamplingDelay_12Cycles ((uint32_t)0x00000700)
```

ADC_TwoSamplingDelay_13Cycles

```
#define ADC_TwoSamplingDelay_13Cycles ((uint32_t)0x00000800)
```

ADC_TwoSamplingDelay_14Cycles

```
#define ADC_TwoSamplingDelay_14Cycles ((uint32_t)0x00000900)
```


ADC_TwoSamplingDelay_15Cycles

```
#define ADC_TwoSamplingDelay_15Cycles ((uint32_t)0x00000A00)
```

ADC_TwoSamplingDelay_16Cycles

```
#define ADC_TwoSamplingDelay_16Cycles ((uint32_t)0x00000B00)
```

ADC_TwoSamplingDelay_17Cycles

```
#define ADC_TwoSamplingDelay_17Cycles ((uint32_t)0x00000C00)
```

ADC_TwoSamplingDelay_18Cycles

```
#define ADC_TwoSamplingDelay_18Cycles ((uint32_t)0x00000D00)
```

ADC_TwoSamplingDelay_19Cycles

```
#define ADC_TwoSamplingDelay_19Cycles ((uint32_t)0x00000E00)
```

ADC_TwoSamplingDelay_20Cycles

```
#define ADC_TwoSamplingDelay_20Cycles ((uint32_t)0x00000F00)
```

ADC_TwoSamplingDelay_5Cycles

```
#define ADC_TwoSamplingDelay_5Cycles ((uint32_t)0x00000000)
```

ADC_TwoSamplingDelay_6Cycles

```
#define ADC_TwoSamplingDelay_6Cycles ((uint32_t)0x00000100)
```

ADC_TwoSamplingDelay_7Cycles

```
#define ADC_TwoSamplingDelay_7Cycles ((uint32_t)0x00000200)
```

ADC_TwoSamplingDelay_8Cycles

```
#define ADC_TwoSamplingDelay_8Cycles ((uint32_t)0x00000300)
```

ADC_TwoSamplingDelay_9Cycles

```
#define ADC_TwoSamplingDelay_9Cycles ((uint32_t)0x00000400)
```

IS_ADC_SAMPLING_DELAY

```
#define IS_ADC_SAMPLING_DELAY(  
    DELAY )
```

Value:

```
((DELAY) == ADC_TwoSamplingDelay_5Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_6Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_7Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_8Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_9Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_10Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_11Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_12Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_13Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_14Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_15Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_16Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_17Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_18Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_19Cycles) || \  
(DELAY) == ADC_TwoSamplingDelay_20Cycles))
```

4.1.2.5.7 ADC_resolution**Macros**

- `#define ADC_Resolution_12b ((uint32_t)0x00000000)`
- `#define ADC_Resolution_10b ((uint32_t)0x01000000)`
- `#define ADC_Resolution_8b ((uint32_t)0x02000000)`
- `#define ADC_Resolution_6b ((uint32_t)0x03000000)`
- `#define IS_ADC_RESOLUTION(RESOLUTION)`

4.1.2.5.7.1 Detailed Description**4.1.2.5.7.2 Macro Definition Documentation****ADC_Resolution_10b**

```
#define ADC_Resolution_10b ((uint32_t)0x01000000)
```

ADC_Resolution_12b

```
#define ADC_Resolution_12b ((uint32_t)0x00000000)
```

ADC_Resolution_6b

```
#define ADC_Resolution_6b ((uint32_t)0x03000000)
```

ADC_Resolution_8b

```
#define ADC_Resolution_8b ((uint32_t)0x02000000)
```

IS_ADC_RESOLUTION

```
#define IS_ADC_RESOLUTION(  
    RESOLUTION )
```

Value:

```
((RESOLUTION) == ADC_Resolution_12b) || \  
((RESOLUTION) == ADC_Resolution_10b) || \  
((RESOLUTION) == ADC_Resolution_8b) || \  
((RESOLUTION) == ADC_Resolution_6b)
```

4.1.2.5.8 ADC_external_trigger_edge_for_regular_channels_conversion**Macros**

- #define [ADC_ExternalTrigConvEdge_None](#) ((uint32_t)0x00000000)
- #define [ADC_ExternalTrigConvEdge_Rising](#) ((uint32_t)0x10000000)
- #define [ADC_ExternalTrigConvEdge_Falling](#) ((uint32_t)0x20000000)
- #define [ADC_ExternalTrigConvEdge_RisingFalling](#) ((uint32_t)0x30000000)
- #define [IS_ADC_EXT_TRIG_EDGE](#)(EDGE)

4.1.2.5.8.1 Detailed Description**4.1.2.5.8.2 Macro Definition Documentation****ADC_ExternalTrigConvEdge_Falling**

```
#define ADC_ExternalTrigConvEdge_Falling ((uint32_t)0x20000000)
```

ADC_ExternalTrigConvEdge_None

```
#define ADC_ExternalTrigConvEdge_None ((uint32_t)0x00000000)
```

ADC_ExternalTrigConvEdge_Rising

```
#define ADC_ExternalTrigConvEdge_Rising ((uint32_t)0x10000000)
```

ADC_ExternalTrigConvEdge_RisingFalling

```
#define ADC_ExternalTrigConvEdge_RisingFalling ((uint32_t)0x30000000)
```

IS_ADC_EXT_TRIG_EDGE

```
#define IS_ADC_EXT_TRIG_EDGE(  
    EDGE )
```

Value:

```
(( (EDGE) == ADC_ExternalTrigConvEdge_None) || \  
( (EDGE) == ADC_ExternalTrigConvEdge_Rising) || \  
( (EDGE) == ADC_ExternalTrigConvEdge_Falling) || \  
( (EDGE) == ADC_ExternalTrigConvEdge_RisingFalling))
```

4.1.2.5.9 ADC_extrenal_trigger_sources_for_regular_channels_conversion

Macros

- #define [ADC_ExternalTrigConv_T1_CC1](#) ((uint32_t)0x00000000)
- #define [ADC_ExternalTrigConv_T1_CC2](#) ((uint32_t)0x01000000)
- #define [ADC_ExternalTrigConv_T1_CC3](#) ((uint32_t)0x02000000)
- #define [ADC_ExternalTrigConv_T2_CC2](#) ((uint32_t)0x03000000)
- #define [ADC_ExternalTrigConv_T2_CC3](#) ((uint32_t)0x04000000)
- #define [ADC_ExternalTrigConv_T2_CC4](#) ((uint32_t)0x05000000)
- #define [ADC_ExternalTrigConv_T2_TRGO](#) ((uint32_t)0x06000000)
- #define [ADC_ExternalTrigConv_T3_CC1](#) ((uint32_t)0x07000000)
- #define [ADC_ExternalTrigConv_T3_TRGO](#) ((uint32_t)0x08000000)
- #define [ADC_ExternalTrigConv_T4_CC4](#) ((uint32_t)0x09000000)
- #define [ADC_ExternalTrigConv_T5_CC1](#) ((uint32_t)0x0A000000)
- #define [ADC_ExternalTrigConv_T5_CC2](#) ((uint32_t)0x0B000000)
- #define [ADC_ExternalTrigConv_T5_CC3](#) ((uint32_t)0x0C000000)
- #define [ADC_ExternalTrigConv_T8_CC1](#) ((uint32_t)0x0D000000)
- #define [ADC_ExternalTrigConv_T8_TRGO](#) ((uint32_t)0x0E000000)
- #define [ADC_ExternalTrigConv_Ext_IT11](#) ((uint32_t)0x0F000000)
- #define [IS_ADC_EXT_TRIG](#)(REGTRIG)

4.1.2.5.9.1 Detailed Description

4.1.2.5.9.2 Macro Definition Documentation

ADC_ExternalTrigConv_Ext_IT11

```
#define ADC_ExternalTrigConv_Ext_IT11 ((uint32_t)0x0F000000)
```

ADC_ExternalTrigConv_T1_CC1

```
#define ADC_ExternalTrigConv_T1_CC1 ((uint32_t)0x00000000)
```

ADC_ExternalTrigConv_T1_CC2

```
#define ADC_ExternalTrigConv_T1_CC2 ((uint32_t)0x01000000)
```

ADC_ExternalTrigConv_T1_CC3

```
#define ADC_ExternalTrigConv_T1_CC3 ((uint32_t)0x02000000)
```

ADC_ExternalTrigConv_T2_CC2

```
#define ADC_ExternalTrigConv_T2_CC2 ((uint32_t)0x03000000)
```

ADC_ExternalTrigConv_T2_CC3

```
#define ADC_ExternalTrigConv_T2_CC3 ((uint32_t)0x04000000)
```

ADC_ExternalTrigConv_T2_CC4

```
#define ADC_ExternalTrigConv_T2_CC4 ((uint32_t)0x05000000)
```

ADC_ExternalTrigConv_T2_TRGO

```
#define ADC_ExternalTrigConv_T2_TRGO ((uint32_t)0x06000000)
```

ADC_ExternalTrigConv_T3_CC1

```
#define ADC_ExternalTrigConv_T3_CC1 ((uint32_t)0x07000000)
```

ADC_ExternalTrigConv_T3_TRGO

```
#define ADC_ExternalTrigConv_T3_TRGO ((uint32_t)0x08000000)
```

ADC_ExternalTrigConv_T4_CC4

```
#define ADC_ExternalTrigConv_T4_CC4 ((uint32_t)0x09000000)
```

ADC_ExternalTrigConv_T5_CC1

```
#define ADC_ExternalTrigConv_T5_CC1 ((uint32_t)0x0A000000)
```

ADC_ExternalTrigConv_T5_CC2

```
#define ADC_ExternalTrigConv_T5_CC2 ((uint32_t)0x0B000000)
```

ADC_ExternalTrigConv_T5_CC3

```
#define ADC_ExternalTrigConv_T5_CC3 ((uint32_t)0x0C000000)
```

ADC_ExternalTrigConv_T8_CC1

```
#define ADC_ExternalTrigConv_T8_CC1 ((uint32_t)0x0D000000)
```

ADC_ExternalTrigConv_T8_TRGO

```
#define ADC_ExternalTrigConv_T8_TRGO ((uint32_t)0x0E000000)
```

IS_ADC_EXT_TRIG

```
#define IS_ADC_EXT_TRIG(  
    REGTRIG )
```

Value:

```
((REGTRIG) == ADC_ExternalTrigConv_T1_CC1) || \  
((REGTRIG) == ADC_ExternalTrigConv_T1_CC2) || \  
((REGTRIG) == ADC_ExternalTrigConv_T1_CC3) || \  
((REGTRIG) == ADC_ExternalTrigConv_T2_CC2) || \  
((REGTRIG) == ADC_ExternalTrigConv_T2_CC3) || \  
((REGTRIG) == ADC_ExternalTrigConv_T2_CC4) || \  
((REGTRIG) == ADC_ExternalTrigConv_T2_TRGO) || \  
((REGTRIG) == ADC_ExternalTrigConv_T3_CC1) || \  
((REGTRIG) == ADC_ExternalTrigConv_T3_TRGO) || \  
((REGTRIG) == ADC_ExternalTrigConv_T4_CC4) || \  
((REGTRIG) == ADC_ExternalTrigConv_T5_CC1) || \  
((REGTRIG) == ADC_ExternalTrigConv_T5_CC2) || \  
((REGTRIG) == ADC_ExternalTrigConv_T5_CC3) || \  
((REGTRIG) == ADC_ExternalTrigConv_T8_CC1) || \  
((REGTRIG) == ADC_ExternalTrigConv_T8_TRGO) || \  
((REGTRIG) == ADC_ExternalTrigConv_Ext_IT11))
```

4.1.2.5.10 ADC_data_align**Macros**

- #define [ADC_DataAlign_Right](#) ((uint32_t)0x00000000)
- #define [ADC_DataAlign_Left](#) ((uint32_t)0x00000800)
- #define [IS_ADC_DATA_ALIGN](#)(ALIGN)

4.1.2.5.10.1 Detailed Description**4.1.2.5.10.2 Macro Definition Documentation****ADC_DataAlign_Left**

```
#define ADC_DataAlign_Left ((uint32_t)0x00000800)
```

ADC_DataAlign_Right

```
#define ADC_DataAlign_Right ((uint32_t)0x00000000)
```

IS_ADC_DATA_ALIGN

```
#define IS_ADC_DATA_ALIGN(  
    ALIGN )
```

Value:

```
((ALIGN) == ADC_DataAlign_Right) || \  
((ALIGN) == ADC_DataAlign_Left))
```

4.1.2.5.11 ADC_channels**Macros**

- #define [ADC_Channel_0](#) ((uint8_t)0x00)
- #define [ADC_Channel_1](#) ((uint8_t)0x01)
- #define [ADC_Channel_2](#) ((uint8_t)0x02)
- #define [ADC_Channel_3](#) ((uint8_t)0x03)
- #define [ADC_Channel_4](#) ((uint8_t)0x04)
- #define [ADC_Channel_5](#) ((uint8_t)0x05)
- #define [ADC_Channel_6](#) ((uint8_t)0x06)
- #define [ADC_Channel_7](#) ((uint8_t)0x07)
- #define [ADC_Channel_8](#) ((uint8_t)0x08)
- #define [ADC_Channel_9](#) ((uint8_t)0x09)
- #define [ADC_Channel_10](#) ((uint8_t)0x0A)
- #define [ADC_Channel_11](#) ((uint8_t)0x0B)
- #define [ADC_Channel_12](#) ((uint8_t)0x0C)
- #define [ADC_Channel_13](#) ((uint8_t)0x0D)
- #define [ADC_Channel_14](#) ((uint8_t)0x0E)
- #define [ADC_Channel_15](#) ((uint8_t)0x0F)
- #define [ADC_Channel_16](#) ((uint8_t)0x10)
- #define [ADC_Channel_17](#) ((uint8_t)0x11)
- #define [ADC_Channel_18](#) ((uint8_t)0x12)
- #define [ADC_Channel_TempSensor](#) ((uint8_t)[ADC_Channel_16](#))
- #define [ADC_Channel_Vrefint](#) ((uint8_t)[ADC_Channel_17](#))
- #define [ADC_Channel_Vbat](#) ((uint8_t)[ADC_Channel_18](#))
- #define [IS_ADC_CHANNEL](#)(CHANNEL)

4.1.2.5.11.1 Detailed Description**4.1.2.5.11.2 Macro Definition Documentation****ADC_Channel_0**

```
#define ADC_Channel_0 ((uint8_t)0x00)
```

ADC_Channel_1

```
#define ADC_Channel_1 ((uint8_t)0x01)
```

ADC_Channel_10

```
#define ADC_Channel_10 ((uint8_t)0x0A)
```

ADC_Channel_11

```
#define ADC_Channel_11 ((uint8_t)0x0B)
```

ADC_Channel_12

```
#define ADC_Channel_12 ((uint8_t)0x0C)
```

ADC_Channel_13

```
#define ADC_Channel_13 ((uint8_t)0x0D)
```

ADC_Channel_14

```
#define ADC_Channel_14 ((uint8_t)0x0E)
```

ADC_Channel_15

```
#define ADC_Channel_15 ((uint8_t)0x0F)
```

ADC_Channel_16

```
#define ADC_Channel_16 ((uint8_t)0x10)
```

ADC_Channel_17

```
#define ADC_Channel_17 ((uint8_t)0x11)
```

ADC_Channel_18

```
#define ADC_Channel_18 ((uint8_t)0x12)
```


ADC_Channel_2

```
#define ADC_Channel_2 ((uint8_t)0x02)
```

ADC_Channel_3

```
#define ADC_Channel_3 ((uint8_t)0x03)
```

ADC_Channel_4

```
#define ADC_Channel_4 ((uint8_t)0x04)
```

ADC_Channel_5

```
#define ADC_Channel_5 ((uint8_t)0x05)
```

ADC_Channel_6

```
#define ADC_Channel_6 ((uint8_t)0x06)
```

ADC_Channel_7

```
#define ADC_Channel_7 ((uint8_t)0x07)
```

ADC_Channel_8

```
#define ADC_Channel_8 ((uint8_t)0x08)
```

ADC_Channel_9

```
#define ADC_Channel_9 ((uint8_t)0x09)
```

ADC_Channel_TempSensor

```
#define ADC_Channel_TempSensor ((uint8_t)ADC_Channel_16)
```

ADC_Channel_Vbat

```
#define ADC_Channel_Vbat ((uint8_t)ADC_Channel_18)
```

ADC_Channel_Vrefint

```
#define ADC_Channel_Vrefint ((uint8_t)ADC_Channel_17)
```

IS_ADC_CHANNEL

```
#define IS_ADC_CHANNEL(  
    CHANNEL )
```

Value:

```
((CHANNEL) == ADC_Channel_0) || \  
((CHANNEL) == ADC_Channel_1) || \  
((CHANNEL) == ADC_Channel_2) || \  
((CHANNEL) == ADC_Channel_3) || \  
((CHANNEL) == ADC_Channel_4) || \  
((CHANNEL) == ADC_Channel_5) || \  
((CHANNEL) == ADC_Channel_6) || \  
((CHANNEL) == ADC_Channel_7) || \  
((CHANNEL) == ADC_Channel_8) || \  
((CHANNEL) == ADC_Channel_9) || \  
((CHANNEL) == ADC_Channel_10) || \  
((CHANNEL) == ADC_Channel_11) || \  
((CHANNEL) == ADC_Channel_12) || \  
((CHANNEL) == ADC_Channel_13) || \  
((CHANNEL) == ADC_Channel_14) || \  
((CHANNEL) == ADC_Channel_15) || \  
((CHANNEL) == ADC_Channel_16) || \  
((CHANNEL) == ADC_Channel_17) || \  
((CHANNEL) == ADC_Channel_18))
```

4.1.2.5.12 ADC_sampling_times**Macros**

- #define [ADC_SampleTime_3Cycles](#) ((uint8_t)0x00)
- #define [ADC_SampleTime_15Cycles](#) ((uint8_t)0x01)
- #define [ADC_SampleTime_28Cycles](#) ((uint8_t)0x02)
- #define [ADC_SampleTime_56Cycles](#) ((uint8_t)0x03)
- #define [ADC_SampleTime_84Cycles](#) ((uint8_t)0x04)
- #define [ADC_SampleTime_112Cycles](#) ((uint8_t)0x05)
- #define [ADC_SampleTime_144Cycles](#) ((uint8_t)0x06)
- #define [ADC_SampleTime_480Cycles](#) ((uint8_t)0x07)
- #define [IS_ADC_SAMPLE_TIME](#)(TIME)

4.1.2.5.12.1 Detailed Description**4.1.2.5.12.2 Macro Definition Documentation****ADC_SampleTime_112Cycles**

```
#define ADC_SampleTime_112Cycles ((uint8_t)0x05)
```

ADC_SampleTime_144Cycles

```
#define ADC_SampleTime_144Cycles ((uint8_t)0x06)
```

ADC_SampleTime_15Cycles

```
#define ADC_SampleTime_15Cycles ((uint8_t)0x01)
```

ADC_SampleTime_28Cycles

```
#define ADC_SampleTime_28Cycles ((uint8_t)0x02)
```

ADC_SampleTime_3Cycles

```
#define ADC_SampleTime_3Cycles ((uint8_t)0x00)
```

ADC_SampleTime_480Cycles

```
#define ADC_SampleTime_480Cycles ((uint8_t)0x07)
```

ADC_SampleTime_56Cycles

```
#define ADC_SampleTime_56Cycles ((uint8_t)0x03)
```

ADC_SampleTime_84Cycles

```
#define ADC_SampleTime_84Cycles ((uint8_t)0x04)
```

IS_ADC_SAMPLE_TIME

```
#define IS_ADC_SAMPLE_TIME(  
    TIME )
```

Value:

```
((TIME) == ADC_SampleTime_3Cycles) || \
((TIME) == ADC_SampleTime_15Cycles) || \
((TIME) == ADC_SampleTime_28Cycles) || \
((TIME) == ADC_SampleTime_56Cycles) || \
((TIME) == ADC_SampleTime_84Cycles) || \
((TIME) == ADC_SampleTime_112Cycles) || \
((TIME) == ADC_SampleTime_144Cycles) || \
((TIME) == ADC_SampleTime_480Cycles)
```

4.1.2.5.13 ADC_external_trigger_edge_for_injected_channels_conversion**Macros**

- #define [ADC_ExternalTrigInjecConvEdge_None](#) ((uint32_t)0x00000000)
- #define [ADC_ExternalTrigInjecConvEdge_Rising](#) ((uint32_t)0x00100000)
- #define [ADC_ExternalTrigInjecConvEdge_Falling](#) ((uint32_t)0x00200000)
- #define [ADC_ExternalTrigInjecConvEdge_RisingFalling](#) ((uint32_t)0x00300000)
- #define [IS_ADC_EXT_INJEC_TRIG_EDGE](#)(EDGE)

4.1.2.5.13.1 Detailed Description

4.1.2.5.13.2 Macro Definition Documentation

ADC_ExternalTrigInjecConvEdge_Falling

```
#define ADC_ExternalTrigInjecConvEdge_Falling ((uint32_t)0x00200000)
```

ADC_ExternalTrigInjecConvEdge_None

```
#define ADC_ExternalTrigInjecConvEdge_None ((uint32_t)0x00000000)
```

ADC_ExternalTrigInjecConvEdge_Rising

```
#define ADC_ExternalTrigInjecConvEdge_Rising ((uint32_t)0x00100000)
```

ADC_ExternalTrigInjecConvEdge_RisingFalling

```
#define ADC_ExternalTrigInjecConvEdge_RisingFalling ((uint32_t)0x00300000)
```

IS_ADC_EXT_INJEC_TRIG_EDGE

```
#define IS_ADC_EXT_INJEC_TRIG_EDGE(  
    EDGE )
```

Value:

```
((EDGE) == ADC_ExternalTrigInjecConvEdge_None) || \  
(EDGE) == ADC_ExternalTrigInjecConvEdge_Rising) || \  
(EDGE) == ADC_ExternalTrigInjecConvEdge_Falling) || \  
(EDGE) == ADC_ExternalTrigInjecConvEdge_RisingFalling)
```

4.1.2.5.14 ADC_extrenal_trigger_sources_for_injected_channels_conversion

Macros

- #define ADC_ExternalTrigInjecConv_T1_CC4 ((uint32_t)0x00000000)
- #define ADC_ExternalTrigInjecConv_T1_TRGO ((uint32_t)0x00010000)
- #define ADC_ExternalTrigInjecConv_T2_CC1 ((uint32_t)0x00020000)
- #define ADC_ExternalTrigInjecConv_T2_TRGO ((uint32_t)0x00030000)
- #define ADC_ExternalTrigInjecConv_T3_CC2 ((uint32_t)0x00040000)
- #define ADC_ExternalTrigInjecConv_T3_CC4 ((uint32_t)0x00050000)
- #define ADC_ExternalTrigInjecConv_T4_CC1 ((uint32_t)0x00060000)
- #define ADC_ExternalTrigInjecConv_T4_CC2 ((uint32_t)0x00070000)
- #define ADC_ExternalTrigInjecConv_T4_CC3 ((uint32_t)0x00080000)
- #define ADC_ExternalTrigInjecConv_T4_TRGO ((uint32_t)0x00090000)
- #define ADC_ExternalTrigInjecConv_T5_CC4 ((uint32_t)0x000A0000)
- #define ADC_ExternalTrigInjecConv_T5_TRGO ((uint32_t)0x000B0000)
- #define ADC_ExternalTrigInjecConv_T8_CC2 ((uint32_t)0x000C0000)
- #define ADC_ExternalTrigInjecConv_T8_CC3 ((uint32_t)0x000D0000)
- #define ADC_ExternalTrigInjecConv_T8_CC4 ((uint32_t)0x000E0000)
- #define ADC_ExternalTrigInjecConv_Ext_IT15 ((uint32_t)0x000F0000)
- #define IS_ADC_EXT_INJEC_TRIG(INJTRIG)

4.1.2.5.14.1 Detailed Description

4.1.2.5.14.2 Macro Definition Documentation

ADC_ExternalTrigInjecConv_Ext_IT15

```
#define ADC_ExternalTrigInjecConv_Ext_IT15 ((uint32_t)0x000F0000)
```

ADC_ExternalTrigInjecConv_T1_CC4

```
#define ADC_ExternalTrigInjecConv_T1_CC4 ((uint32_t)0x00000000)
```

ADC_ExternalTrigInjecConv_T1_TRGO

```
#define ADC_ExternalTrigInjecConv_T1_TRGO ((uint32_t)0x00010000)
```

ADC_ExternalTrigInjecConv_T2_CC1

```
#define ADC_ExternalTrigInjecConv_T2_CC1 ((uint32_t)0x00020000)
```

ADC_ExternalTrigInjecConv_T2_TRGO

```
#define ADC_ExternalTrigInjecConv_T2_TRGO ((uint32_t)0x00030000)
```

ADC_ExternalTrigInjecConv_T3_CC2

```
#define ADC_ExternalTrigInjecConv_T3_CC2 ((uint32_t)0x00040000)
```

ADC_ExternalTrigInjecConv_T3_CC4

```
#define ADC_ExternalTrigInjecConv_T3_CC4 ((uint32_t)0x00050000)
```

ADC_ExternalTrigInjecConv_T4_CC1

```
#define ADC_ExternalTrigInjecConv_T4_CC1 ((uint32_t)0x00060000)
```

ADC_ExternalTrigInjecConv_T4_CC2

```
#define ADC_ExternalTrigInjecConv_T4_CC2 ((uint32_t)0x00070000)
```

ADC_ExternalTrigInjecConv_T4_CC3

```
#define ADC_ExternalTrigInjecConv_T4_CC3 ((uint32_t)0x00080000)
```

ADC_ExternalTrigInjecConv_T4_TRGO

```
#define ADC_ExternalTrigInjecConv_T4_TRGO ((uint32_t)0x00090000)
```

ADC_ExternalTrigInjecConv_T5_CC4

```
#define ADC_ExternalTrigInjecConv_T5_CC4 ((uint32_t)0x000A0000)
```

ADC_ExternalTrigInjecConv_T5_TRGO

```
#define ADC_ExternalTrigInjecConv_T5_TRGO ((uint32_t)0x000B0000)
```

ADC_ExternalTrigInjecConv_T8_CC2

```
#define ADC_ExternalTrigInjecConv_T8_CC2 ((uint32_t)0x000C0000)
```

ADC_ExternalTrigInjecConv_T8_CC3

```
#define ADC_ExternalTrigInjecConv_T8_CC3 ((uint32_t)0x000D0000)
```

ADC_ExternalTrigInjecConv_T8_CC4

```
#define ADC_ExternalTrigInjecConv_T8_CC4 ((uint32_t)0x000E0000)
```

IS_ADC_EXT_INJEC_TRIG

```
#define IS_ADC_EXT_INJEC_TRIG(  
    INJTRIG )
```

Value:

```
((INJTRIG) == ADC_ExternalTrigInjecConv_T1_CC4) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T1_TRGO) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T2_CC1) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T2_TRGO) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T3_CC2) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T3_CC4) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T4_CC1) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T4_CC2) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T4_CC3) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T4_TRGO) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T5_CC4) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T5_TRGO) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T8_CC2) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T8_CC3) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_T8_CC4) || \  
((INJTRIG) == ADC_ExternalTrigInjecConv_Ext_IT15))
```

4.1.2.5.15 ADC_injected_channel_selection

Macros

- #define `ADC_InjectedChannel_1` ((uint8_t)0x14)
- #define `ADC_InjectedChannel_2` ((uint8_t)0x18)
- #define `ADC_InjectedChannel_3` ((uint8_t)0x1C)
- #define `ADC_InjectedChannel_4` ((uint8_t)0x20)
- #define `IS_ADC_INJECTED_CHANNEL`(CHANNEL)

4.1.2.5.15.1 Detailed Description

4.1.2.5.15.2 Macro Definition Documentation

ADC_InjectedChannel_1

```
#define ADC_InjectedChannel_1 ((uint8_t)0x14)
```

ADC_InjectedChannel_2

```
#define ADC_InjectedChannel_2 ((uint8_t)0x18)
```

ADC_InjectedChannel_3

```
#define ADC_InjectedChannel_3 ((uint8_t)0x1C)
```

ADC_InjectedChannel_4

```
#define ADC_InjectedChannel_4 ((uint8_t)0x20)
```

IS_ADC_INJECTED_CHANNEL

```
#define IS_ADC_INJECTED_CHANNEL(  
    CHANNEL )
```

Value:

```
((CHANNEL) == ADC_InjectedChannel_1) || \  
((CHANNEL) == ADC_InjectedChannel_2) || \  
((CHANNEL) == ADC_InjectedChannel_3) || \  
((CHANNEL) == ADC_InjectedChannel_4)
```

4.1.2.5.16 ADC_analog_watchdog_selection

Macros

- `#define ADC_AnalogWatchdog_SingleRegEnable ((uint32_t)0x00800200)`
- `#define ADC_AnalogWatchdog_SingleInjecEnable ((uint32_t)0x00400200)`
- `#define ADC_AnalogWatchdog_SingleRegOrInjecEnable ((uint32_t)0x00C00200)`
- `#define ADC_AnalogWatchdog_AllRegEnable ((uint32_t)0x00800000)`
- `#define ADC_AnalogWatchdog_AllInjecEnable ((uint32_t)0x00400000)`
- `#define ADC_AnalogWatchdog_AllRegAllInjecEnable ((uint32_t)0x00C00000)`
- `#define ADC_AnalogWatchdog_None ((uint32_t)0x00000000)`
- `#define IS_ADC_ANALOG_WATCHDOG(WATCHDOG)`

4.1.2.5.16.1 Detailed Description

4.1.2.5.16.2 Macro Definition Documentation

ADC_AnalogWatchdog_AllInjecEnable

```
#define ADC_AnalogWatchdog_AllInjecEnable ((uint32_t)0x00400000)
```

ADC_AnalogWatchdog_AllRegAllInjecEnable

```
#define ADC_AnalogWatchdog_AllRegAllInjecEnable ((uint32_t)0x00C00000)
```

ADC_AnalogWatchdog_AllRegEnable

```
#define ADC_AnalogWatchdog_AllRegEnable ((uint32_t)0x00800000)
```

ADC_AnalogWatchdog_None

```
#define ADC_AnalogWatchdog_None ((uint32_t)0x00000000)
```

ADC_AnalogWatchdog_SingleInjecEnable

```
#define ADC_AnalogWatchdog_SingleInjecEnable ((uint32_t)0x00400200)
```

ADC_AnalogWatchdog_SingleRegEnable

```
#define ADC_AnalogWatchdog_SingleRegEnable ((uint32_t)0x00800200)
```

ADC_AnalogWatchdog_SingleRegOrInjecEnable

```
#define ADC_AnalogWatchdog_SingleRegOrInjecEnable ((uint32_t)0x00C00200)
```


IS_ADC_ANALOG_WATCHDOG

```
#define IS_ADC_ANALOG_WATCHDOG(
    WATCHDOG )
```

Value:

```
((WATCHDOG) == ADC_AnalogWatchdog_SingleRegEnable) || \
(WATCHDOG) == ADC_AnalogWatchdog_SingleInjecEnable) || \
(WATCHDOG) == ADC_AnalogWatchdog_SingleRegOrInjecEnable) || \
(WATCHDOG) == ADC_AnalogWatchdog_AllRegEnable) || \
(WATCHDOG) == ADC_AnalogWatchdog_AllInjecEnable) || \
(WATCHDOG) == ADC_AnalogWatchdog_AllRegAllInjecEnable) || \
(WATCHDOG) == ADC_AnalogWatchdog_None)
```

4.1.2.5.17 ADC_interrupts_definition**Macros**

- #define **ADC_IT_EOC** ((uint16_t)0x0205)
- #define **ADC_IT_AWD** ((uint16_t)0x0106)
- #define **ADC_IT_JEOC** ((uint16_t)0x0407)
- #define **ADC_IT_OVR** ((uint16_t)0x201A)
- #define **IS_ADC_IT**(IT)

4.1.2.5.17.1 Detailed Description**4.1.2.5.17.2 Macro Definition Documentation****ADC_IT_AWD**

```
#define ADC_IT_AWD ((uint16_t)0x0106)
```

ADC_IT_EOC

```
#define ADC_IT_EOC ((uint16_t)0x0205)
```

ADC_IT_JEOC

```
#define ADC_IT_JEOC ((uint16_t)0x0407)
```

ADC_IT_OVR

```
#define ADC_IT_OVR ((uint16_t)0x201A)
```

IS_ADC_IT

```
#define IS_ADC_IT(
    IT )
```

Value:

```
((IT) == ADC_IT_EOC) || ((IT) == ADC_IT_AWD) || \
((IT) == ADC_IT_JEOC) || ((IT) == ADC_IT_OVR)
```

4.1.2.5.18 ADC_flags_definition

Macros

- #define `ADC_FLAG_AWD` ((uint8_t)0x01)
- #define `ADC_FLAG_EOC` ((uint8_t)0x02)
- #define `ADC_FLAG_JEOC` ((uint8_t)0x04)
- #define `ADC_FLAG_JSTRT` ((uint8_t)0x08)
- #define `ADC_FLAG_STRT` ((uint8_t)0x10)
- #define `ADC_FLAG_OVR` ((uint8_t)0x20)
- #define `IS_ADC_CLEAR_FLAG`(FLAG) (((FLAG) & (uint8_t)0xC0) == 0x00) && ((FLAG) != 0x00)
- #define `IS_ADC_GET_FLAG`(FLAG)

4.1.2.5.18.1 Detailed Description

4.1.2.5.18.2 Macro Definition Documentation

ADC_FLAG_AWD

```
#define ADC_FLAG_AWD ((uint8_t)0x01)
```

ADC_FLAG_EOC

```
#define ADC_FLAG_EOC ((uint8_t)0x02)
```

ADC_FLAG_JEOC

```
#define ADC_FLAG_JEOC ((uint8_t)0x04)
```

ADC_FLAG_JSTRT

```
#define ADC_FLAG_JSTRT ((uint8_t)0x08)
```

ADC_FLAG_OVR

```
#define ADC_FLAG_OVR ((uint8_t)0x20)
```

ADC_FLAG_STRT

```
#define ADC_FLAG_STRT ((uint8_t)0x10)
```

IS_ADC_CLEAR_FLAG

```
#define IS_ADC_CLEAR_FLAG(  
    FLAG ) (((FLAG) & (uint8_t)0xC0) == 0x00) && ((FLAG) != 0x00)
```

IS_ADC_GET_FLAG

```
#define IS_ADC_GET_FLAG(  
    FLAG )
```

Value:

```
((FLAG) == ADC_FLAG_AWD) || \  
(FLAG) == ADC_FLAG_EOC) || \  
(FLAG) == ADC_FLAG_JEOC) || \  
(FLAG) == ADC_FLAG_JSTRT) || \  
(FLAG) == ADC_FLAG_STRT) || \  
(FLAG) == ADC_FLAG_OVR)
```

4.1.2.5.19 ADC_thresholds**Macros**

- #define [IS_ADC_THRESHOLD](#)(THRESHOLD) ((THRESHOLD) <= 0xFFF)

4.1.2.5.19.1 Detailed Description**4.1.2.5.19.2 Macro Definition Documentation****IS_ADC_THRESHOLD**

```
#define IS_ADC_THRESHOLD(  
    THRESHOLD ) ((THRESHOLD) <= 0xFFF)
```

4.1.2.5.20 ADC_injected_offset**Macros**

- #define [IS_ADC_OFFSET](#)(OFFSET) ((OFFSET) <= 0xFFF)

4.1.2.5.20.1 Detailed Description**4.1.2.5.20.2 Macro Definition Documentation****IS_ADC_OFFSET**

```
#define IS_ADC_OFFSET(  
    OFFSET ) ((OFFSET) <= 0xFFF)
```

4.1.2.5.21 ADC_injected_length**Macros**

- #define [IS_ADC_INJECTED_LENGTH](#)(LENGTH) (((LENGTH) >= 0x1) && ((LENGTH) <= 0x4))

4.1.2.5.21.1 Detailed Description

4.1.2.5.21.2 Macro Definition Documentation

IS_ADC_INJECTED_LENGTH

```
#define IS_ADC_INJECTED_LENGTH(  
    LENGTH ) (((LENGTH) >= 0x1) && ((LENGTH) <= 0x4))
```

4.1.2.5.22 ADC_injected_rank

Macros

- #define [IS_ADC_INJECTED_RANK](#)(RANK) (((RANK) >= 0x1) && ((RANK) <= 0x4))

4.1.2.5.22.1 Detailed Description

4.1.2.5.22.2 Macro Definition Documentation

IS_ADC_INJECTED_RANK

```
#define IS_ADC_INJECTED_RANK(  
    RANK ) (((RANK) >= 0x1) && ((RANK) <= 0x4))
```

4.1.2.5.23 ADC_regular_length

Macros

- #define [IS_ADC_REGULAR_LENGTH](#)(LENGTH) (((LENGTH) >= 0x1) && ((LENGTH) <= 0x10))

4.1.2.5.23.1 Detailed Description

4.1.2.5.23.2 Macro Definition Documentation

IS_ADC_REGULAR_LENGTH

```
#define IS_ADC_REGULAR_LENGTH(  
    LENGTH ) (((LENGTH) >= 0x1) && ((LENGTH) <= 0x10))
```

4.1.2.5.24 ADC_regular_rank

Macros

- #define [IS_ADC_REGULAR_RANK](#)(RANK) (((RANK) >= 0x1) && ((RANK) <= 0x10))

4.1.2.5.24.1 Detailed Description**4.1.2.5.24.2 Macro Definition Documentation****IS_ADC_REGULAR_RANK**

```
#define IS_ADC_REGULAR_RANK(  
    RANK ) (((RANK) >= 0x1) && ((RANK) <= 0x10))
```

4.1.2.5.25 ADC_regular_discontinuous_mode_number**Macros**

- `#define IS_ADC_REGULAR_DISC_NUMBER(NUMBER) (((NUMBER) >= 0x1) && ((NUMBER) <= 0x8))`

4.1.2.5.25.1 Detailed Description**4.1.2.5.25.2 Macro Definition Documentation****IS_ADC_REGULAR_DISC_NUMBER**

```
#define IS_ADC_REGULAR_DISC_NUMBER(  
    NUMBER ) (((NUMBER) >= 0x1) && ((NUMBER) <= 0x8))
```

4.1.3 GPIO

GPIO driver modules.

Modules

- [GPIO_Private_Functions](#)
- [GPIO_Exported_Constants](#)

Data Structures

- struct [GPIO_InitTypeDef](#)
GPIO Init structure definition

Macros

- `#define IS_GPIO_ALL_PERIPH(PERIPH)`
- `#define IS_GPIO_MODE(MODE)`
- `#define IS_GPIO_OTYPE(OTYPE) (((OTYPE) == GPIO_OType_PP) || ((OTYPE) == GPIO_OType_OD))`
- `#define IS_GPIO_SPEED(SPEED)`
- `#define IS_GPIO_PUPD(PUPD)`
- `#define IS_GPIO_BIT_ACTION(ACTION) (((ACTION) == Bit_RESET) || ((ACTION) == Bit_SET))`

Enumerations

- enum [GPIO_Mode_TypeDef](#) { [GPIO_Mode_IN](#) = 0x00 , [GPIO_Mode_OUT](#) = 0x01 , [GPIO_Mode_AF](#) = 0x02 , [GPIO_Mode_AN](#) = 0x03 }
- GPIO Configuration Mode enumeration.*
- enum [GPIO_OType_TypeDef](#) { [GPIO_OType_PP](#) = 0x00 , [GPIO_OType_OD](#) = 0x01 }
- GPIO Output type enumeration.*
- enum [GPIO_Speed_TypeDef](#) { [GPIO_Speed_2MHz](#) = 0x00 , [GPIO_Speed_25MHz](#) = 0x01 , [GPIO_Speed_50MHz](#) = 0x02 , [GPIO_Speed_100MHz](#) = 0x03 }
- GPIO Output Maximum frequency enumeration.*
- enum [GPIO_PuPd_TypeDef](#) { [GPIO_PuPd_NOPULL](#) = 0x00 , [GPIO_PuPd_UP](#) = 0x01 , [GPIO_PuPd_DOWN](#) = 0x02 }
- GPIO Configuration PullUp PullDown enumeration.*
- enum [BitAction](#) { [Bit_RESET](#) = 0 , [Bit_SET](#) }
- GPIO Bit SET and Bit RESET enumeration.*

Functions

- void [GPIO_DeInit](#) (GPIO_TypeDef *GPIOx)
- Deinitializes the GPIOx peripheral registers to their default reset values.*
- void [GPIO_Init](#) (GPIO_TypeDef *GPIOx, [GPIO_InitTypeDef](#) *GPIO_InitStruct)
- Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.*
- void [GPIO_StructInit](#) ([GPIO_InitTypeDef](#) *GPIO_InitStruct)
- Fills each GPIO_InitStruct member with its default value.*
- void [GPIO_PinLockConfig](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
- Locks GPIO Pins configuration registers.*
- uint8_t [GPIO_ReadInputDataBit](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
- Reads the specified input port pin.*
- uint16_t [GPIO_ReadInputData](#) (GPIO_TypeDef *GPIOx)
- Reads the specified GPIO input data port.*
- uint8_t [GPIO_ReadOutputDataBit](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
- Reads the specified output data port bit.*
- uint16_t [GPIO_ReadOutputData](#) (GPIO_TypeDef *GPIOx)
- Reads the specified GPIO output data port.*
- void [GPIO_SetBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
- Sets the selected data port bits.*
- void [GPIO_ResetBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
- Clears the selected data port bits.*
- void [GPIO_WriteBit](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, [BitAction](#) BitVal)
- Sets or clears the selected data port bit.*
- void [GPIO_Write](#) (GPIO_TypeDef *GPIOx, uint16_t PortVal)
- Writes data to the specified GPIO data port.*
- void [GPIO_ToggleBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
- Toggles the specified GPIO pins..*
- void [GPIO_PinAFConfig](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_PinSource, uint8_t GPIO_AF)
- Changes the mapping of the specified pin.*

4.1.3.1 Detailed Description

GPIO driver modules.

4.1.3.2 Macro Definition Documentation

4.1.3.2.1 IS_GPIO_ALL_PERIPH

```
#define IS_GPIO_ALL_PERIPH(  
    PERIPH )
```

Value:

```
((PERIPH) == GPIOA) || \  
((PERIPH) == GPIOB) || \  
((PERIPH) == GPIOC) || \  
((PERIPH) == GPIOD) || \  
((PERIPH) == GPIOE) || \  
((PERIPH) == GPIOF) || \  
((PERIPH) == GPIOG) || \  
((PERIPH) == GPIOH) || \  
((PERIPH) == GPIOI)
```

4.1.3.2.2 IS_GPIO_BIT_ACTION

```
#define IS_GPIO_BIT_ACTION(  
    ACTION ) ((ACTION) == Bit_RESET) || ((ACTION) == Bit_SET)
```

4.1.3.2.3 IS_GPIO_MODE

```
#define IS_GPIO_MODE(  
    MODE )
```

Value:

```
((MODE) == GPIO_Mode_IN) || ((MODE) == GPIO_Mode_OUT) || \  
((MODE) == GPIO_Mode_AF) || ((MODE) == GPIO_Mode_AN)
```

4.1.3.2.4 IS_GPIO_OTYPE

```
#define IS_GPIO_OTYPE(  
    OTYPE ) ((OTYPE) == GPIO_OType_PP) || ((OTYPE) == GPIO_OType_OD)
```

4.1.3.2.5 IS_GPIO_PUPD

```
#define IS_GPIO_PUPD(  
    PUPD )
```

Value:

```
((PUPD) == GPIO_PuPd_NOPULL) || ((PUPD) == GPIO_PuPd_UP) || \  
((PUPD) == GPIO_PuPd_DOWN)
```

4.1.3.2.6 IS_GPIO_SPEED

```
#define IS_GPIO_SPEED(  
    SPEED )
```

Value:

```
((SPEED) == GPIO_Speed_2MHz) || ((SPEED) == GPIO_Speed_25MHz) || \  
((SPEED) == GPIO_Speed_50MHz) || ((SPEED) == GPIO_Speed_100MHz)
```

4.1.3.3 Enumeration Type Documentation

4.1.3.3.1 BitAction

```
enum BitAction
```

GPIO Bit SET and Bit RESET enumeration.

Enumerator

Bit_RESET	
Bit_SET	

4.1.3.3.2 GPIOMode_TypeDef

enum [GPIOMode_TypeDef](#)

GPIO Configuration Mode enumeration.

Enumerator

GPIO_Mode_IN	GPIO Input Mode
GPIO_Mode_OUT	GPIO Output Mode
GPIO_Mode_AF	GPIO Alternate function Mode
GPIO_Mode_AN	GPIO Analog Mode

4.1.3.3.3 GPIOOType_TypeDef

enum [GPIOOType_TypeDef](#)

GPIO Output type enumeration.

Enumerator

GPIO_OType_PP	
GPIO_OType_OD	

4.1.3.3.4 GPIOPuPd_TypeDef

enum [GPIOPuPd_TypeDef](#)

GPIO Configuration PullUp PullDown enumeration.

Enumerator

GPIO_PuPd_NOPULL	
GPIO_PuPd_UP	
GPIO_PuPd_DOWN	

4.1.3.3.5 GPIOSpeed_TypeDef

enum [GPIOSpeed_TypeDef](#)

GPIO Output Maximum frequency enumeration.

Enumerator

GPIO_Speed_2MHz	Low speed
GPIO_Speed_25MHz	Medium speed
GPIO_Speed_50MHz	Fast speed
GPIO_Speed_100MHz	High speed on 30 pF (80 MHz Output max speed on 15 pF)

4.1.3.4 Function Documentation

4.1.3.4.1 GPIO_DeInit()

```
void GPIO_DeInit (
    GPIO_TypeDef * GPIOx )
```

Deinitializes the GPIOx peripheral registers to their default reset values.

Note

By default, The GPIO pins are configured in input floating mode (except JTAG pins).

Parameters

GPIOx	where x can be (A..I) to select the GPIO peripheral.
-------	--

Return values

None	
------	--

4.1.3.4.2 GPIO_Init()

```
void GPIO_Init (
    GPIO_TypeDef * GPIOx,
    GPIO_InitTypeDef * GPIO_InitStruct )
```

Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.

Parameters

GPIOx	where x can be (A..I) to select the GPIO peripheral.
GPIO_InitStruct	pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

None	
------	--

4.1.3.4.3 GPIO_PinAFConfig()

```
void GPIO_PinAFConfig (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_PinSource,
    uint8_t GPIO_AF )
```

Changes the mapping of the specified pin.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_PinSource</i>	specifies the pin for the Alternate function. This parameter can be GPIO_PinSourcex where x can be (0..15).

Parameters

<i>GPIO_AFSelection</i>	<p>selects the pin to used as Alternate function. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • GPIO_AF_RTC_50Hz: Connect RTC_50Hz pin to AF0 (default after reset) • GPIO_AF_MCO: Connect MCO pin (MCO1 and MCO2) to AF0 (default after reset) • GPIO_AF_TAMPER: Connect TAMPER pins (TAMPER_1 and TAMPER_2) to AF0 (default after reset) • GPIO_AF_SWJ: Connect SWJ pins (SWD and JTAG)to AF0 (default after reset) • GPIO_AF_TRACE: Connect TRACE pins to AF0 (default after reset) • GPIO_AF_TIM1: Connect TIM1 pins to AF1 • GPIO_AF_TIM2: Connect TIM2 pins to AF1 • GPIO_AF_TIM3: Connect TIM3 pins to AF2 • GPIO_AF_TIM4: Connect TIM4 pins to AF2 • GPIO_AF_TIM5: Connect TIM5 pins to AF2 • GPIO_AF_TIM8: Connect TIM8 pins to AF3 • GPIO_AF_TIM9: Connect TIM9 pins to AF3 • GPIO_AF_TIM10: Connect TIM10 pins to AF3 • GPIO_AF_TIM11: Connect TIM11 pins to AF3 • GPIO_AF_I2C1: Connect I2C1 pins to AF4 • GPIO_AF_I2C2: Connect I2C2 pins to AF4 • GPIO_AF_I2C3: Connect I2C3 pins to AF4 • GPIO_AF_SPI1: Connect SPI1 pins to AF5 • GPIO_AF_SPI2: Connect SPI2/I2S2 pins to AF5 • GPIO_AF_SPI3: Connect SPI3/I2S3 pins to AF6 • GPIO_AF_I2S3ext: Connect I2S3ext pins to AF7 • GPIO_AF_USART1: Connect USART1 pins to AF7 • GPIO_AF_USART2: Connect USART2 pins to AF7 • GPIO_AF_USART3: Connect USART3 pins to AF7 • GPIO_AF_UART4: Connect UART4 pins to AF8 • GPIO_AF_UART5: Connect UART5 pins to AF8 • GPIO_AF_USART6: Connect USART6 pins to AF8 • GPIO_AF_CAN1: Connect CAN1 pins to AF9 • GPIO_AF_CAN2: Connect CAN2 pins to AF9 • GPIO_AF_TIM12: Connect TIM12 pins to AF9 • GPIO_AF_TIM13: Connect TIM13 pins to AF9 • GPIO_AF_TIM14: Connect TIM14 pins to AF9 • GPIO_AF_OTG_FS: Connect OTG_FS pins to AF10
	<ul style="list-style-type: none"> • GPIO_AF_OTG_HS: Connect OTG_HS pins to AF10 • GPIO_AF_ETH: Connect ETHERNET pins to AF11 • GPIO_AF_FSMC: Connect FSMC pins to AF12

Parameters

Return values

<i>None</i>	
-------------	--

4.1.3.4.4 GPIO_PinLockConfig()

```
void GPIO_PinLockConfig (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Locks GPIO Pins configuration registers.

Note

The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.

The configuration of the locked GPIO pins can no longer be modified until the next reset.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

<i>None</i>	
-------------	--

4.1.3.4.5 GPIO_ReadInputData()

```
uint16_t GPIO_ReadInputData (
    GPIO_TypeDef * GPIOx )
```

Reads the specified GPIO input data port.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
--------------	--

Return values

<i>GPIO</i>	input data port value.
-------------	------------------------

4.1.3.4.6 GPIO_ReadInputDataBit()

```
uint8_t GPIO_ReadInputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Reads the specified input port pin.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bit to read. This parameter can be GPIO_Pin_x where x can be (0..15).

Return values

<i>The</i>	input port pin value.
------------	-----------------------

4.1.3.4.7 GPIO_ReadOutputData()

```
uint16_t GPIO_ReadOutputData (
    GPIO_TypeDef * GPIOx )
```

Reads the specified GPIO output data port.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
--------------	--

Return values

<i>GPIO</i>	output data port value.
-------------	-------------------------

4.1.3.4.8 GPIO_ReadOutputDataBit()

```
uint8_t GPIO_ReadOutputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Reads the specified output data port bit.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bit to read. This parameter can be GPIO_Pin_x where x can be (0..15).

Return values

<i>The</i>	output port pin value.
------------	------------------------

4.1.3.4.9 GPIO_ResetBits()

```
void GPIO_ResetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Clears the selected data port bits.

Note

This functions 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.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bits to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

<i>None</i>	
-------------	--

4.1.3.4.10 GPIO_SetBits()

```
void GPIO_SetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Sets the selected data port bits.

Note

This functions 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.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bits to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

<i>None</i>	
-------------	--

4.1.3.4.11 GPIO_StructInit()

```
void GPIO_StructInit (
    GPIO_InitTypeDef * GPIO_InitStruct )
```

Fills each GPIO_InitStruct member with its default value.

Parameters

<i>GPIO_InitStruct</i>	: pointer to a GPIO_InitTypeDef structure which will be initialized.
------------------------	--

Return values

<i>None</i>	
-------------	--

4.1.3.4.12 GPIO_ToggleBits()

```
void GPIO_ToggleBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Toggles the specified GPIO pins..

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	Specifies the pins to be toggled.

Return values

<i>None</i>	
-------------	--

4.1.3.4.13 GPIO_Write()

```
void GPIO_Write (
    GPIO_TypeDef * GPIOx,
    uint16_t PortVal )
```

Writes data to the specified GPIO data port.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>PortVal</i>	specifies the value to be written to the port output data register.

Return values

None	
------	--

4.1.3.4.14 GPIO_WriteBit()

```
void GPIO_WriteBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin,
    BitAction BitVal )
```

Sets or clears the selected data port bit.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bit to be written. This parameter can be one of GPIO_Pin_x where x can be (0..15).
<i>BitVal</i>	specifies the value to be written to the selected bit. This parameter can be one of the BitAction enum values: <ul style="list-style-type: none"> • Bit_RESET: to clear the port pin • Bit_SET: to set the port pin

Return values

None	
------	--

4.1.3.5 GPIO_Private_Functions

Modules

- [Initialization and Configuration](#)
Initialization and Configuration.
- [GPIO Read and Write](#)
GPIO Read and Write.
- [GPIO Alternate functions configuration function](#)
GPIO Alternate functions configuration function.

4.1.3.5.1 Detailed Description

4.1.3.5.2 Initialization and Configuration

Initialization and Configuration.

Functions

- void [GPIO_DeInit](#) (GPIO_TypeDef *GPIOx)
Deinitializes the GPIOx peripheral registers to their default reset values.
- void [GPIO_Init](#) (GPIO_TypeDef *GPIOx, [GPIO_InitTypeDef](#) *GPIO_InitStruct)
Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
- void [GPIO_StructInit](#) ([GPIO_InitTypeDef](#) *GPIO_InitStruct)
Fills each GPIO_InitStruct member with its default value.
- void [GPIO_PinLockConfig](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Locks GPIO Pins configuration registers.

4.1.3.5.2.1 Detailed Description

Initialization and Configuration.

```
=====
Initialization and Configuration
=====
```

4.1.3.5.2.2 Function Documentation

GPIO_DeInit()

```
void GPIO_DeInit (
    GPIO_TypeDef * GPIOx )
```

Deinitializes the GPIOx peripheral registers to their default reset values.

Note

By default, The GPIO pins are configured in input floating mode (except JTAG pins).

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
--------------	--

Return values

<i>None</i>	
-------------	--

GPIO_Init()

```
void GPIO_Init (
    GPIO_TypeDef * GPIOx,
    GPIO\_InitTypeDef * GPIO_InitStruct )
```

Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_InitStruct</i>	pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

<i>None</i>	
-------------	--

GPIO_PinLockConfig()

```
void GPIO_PinLockConfig (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Locks GPIO Pins configuration registers.

Note

The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.

The configuration of the locked GPIO pins can no longer be modified until the next reset.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

<i>None</i>	
-------------	--

GPIO_StructInit()

```
void GPIO_StructInit (
    GPIO_InitTypeDef * GPIO_InitStruct )
```

Fills each GPIO_InitStruct member with its default value.

Parameters

<i>GPIO_InitStruct</i>	: pointer to a GPIO_InitTypeDef structure which will be initialized.
------------------------	--

Return values

None	
------	--

4.1.3.5.3 GPIO Read and Write

GPIO Read and Write.

Functions

- `uint8_t GPIO_ReadInputDataBit` (`GPIO_TypeDef *GPIOx`, `uint16_t GPIO_Pin`)
Reads the specified input port pin.
- `uint16_t GPIO_ReadInputData` (`GPIO_TypeDef *GPIOx`)
Reads the specified GPIO input data port.
- `uint8_t GPIO_ReadOutputDataBit` (`GPIO_TypeDef *GPIOx`, `uint16_t GPIO_Pin`)
Reads the specified output data port bit.
- `uint16_t GPIO_ReadOutputData` (`GPIO_TypeDef *GPIOx`)
Reads the specified GPIO output data port.
- `void GPIO_SetBits` (`GPIO_TypeDef *GPIOx`, `uint16_t GPIO_Pin`)
Sets the selected data port bits.
- `void GPIO_ResetBits` (`GPIO_TypeDef *GPIOx`, `uint16_t GPIO_Pin`)
Clears the selected data port bits.
- `void GPIO_WriteBit` (`GPIO_TypeDef *GPIOx`, `uint16_t GPIO_Pin`, `BitAction BitVal`)
Sets or clears the selected data port bit.
- `void GPIO_Write` (`GPIO_TypeDef *GPIOx`, `uint16_t PortVal`)
Writes data to the specified GPIO data port.
- `void GPIO_ToggleBits` (`GPIO_TypeDef *GPIOx`, `uint16_t GPIO_Pin`)
Toggles the specified GPIO pins..

4.1.3.5.3.1 Detailed Description

GPIO Read and Write.

```
=====
                        GPIO Read and Write
=====
```

4.1.3.5.3.2 Function Documentation

GPIO_ReadInputData()

```
uint16_t GPIO_ReadInputData (
    GPIO_TypeDef * GPIOx )
```

Reads the specified GPIO input data port.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
--------------	--

Return values

<i>GPIO</i>	input data port value.
-------------	------------------------

GPIO_ReadInputDataBit()

```
uint8_t GPIO_ReadInputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Reads the specified input port pin.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bit to read. This parameter can be GPIO_Pin_x where x can be (0..15).

Return values

<i>The</i>	input port pin value.
------------	-----------------------

GPIO_ReadOutputData()

```
uint16_t GPIO_ReadOutputData (
    GPIO_TypeDef * GPIOx )
```

Reads the specified GPIO output data port.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
--------------	--

Return values

<i>GPIO</i>	output data port value.
-------------	-------------------------

GPIO_ReadOutputDataBit()

```
uint8_t GPIO_ReadOutputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Reads the specified output data port bit.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bit to read. This parameter can be GPIO_Pin_x where x can be (0..15).

Return values

<i>The</i>	output port pin value.
------------	------------------------

GPIO_ResetBits()

```
void GPIO_ResetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Clears the selected data port bits.

Note

This functions 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.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bits to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

<i>None</i>	
-------------	--

GPIO_SetBits()

```
void GPIO_SetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Sets the selected data port bits.

Note

This functions 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.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bits to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

<i>None</i>	
-------------	--

GPIO_ToggleBits()

```
void GPIO_ToggleBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

Toggles the specified GPIO pins..

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	Specifies the pins to be toggled.

Return values

<i>None</i>	
-------------	--

GPIO_Write()

```
void GPIO_Write (
    GPIO_TypeDef * GPIOx,
    uint16_t PortVal )
```

Writes data to the specified GPIO data port.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>PortVal</i>	specifies the value to be written to the port output data register.

Return values

<i>None</i>	
-------------	--

GPIO_WriteBit()

```
void GPIO_WriteBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin,
    BitAction BitVal )
```

Sets or clears the selected data port bit.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_Pin</i>	specifies the port bit to be written. This parameter can be one of GPIO_Pin_x where x can be (0..15).
<i>BitVal</i>	specifies the value to be written to the selected bit. This parameter can be one of the BitAction enum values: <ul style="list-style-type: none"> • Bit_RESET: to clear the port pin • Bit_SET: to set the port pin

Return values

<i>None</i>	
-------------	--

4.1.3.5.4 GPIO Alternate functions configuration function

GPIO Alternate functions configuration function.

Functions

- void [GPIO_PinAFConfig](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_PinSource, uint8_t GPIO_AF)
Changes the mapping of the specified pin.

4.1.3.5.4.1 Detailed Description

GPIO Alternate functions configuration function.

```
=====
GPIO Alternate functions configuration function
=====
```

4.1.3.5.4.2 Function Documentation**GPIO_PinAFConfig()**

```
void GPIO_PinAFConfig (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_PinSource,
    uint8_t GPIO_AF )
```

Changes the mapping of the specified pin.

Parameters

<i>GPIOx</i>	where x can be (A..I) to select the GPIO peripheral.
<i>GPIO_PinSource</i>	specifies the pin for the Alternate function. This parameter can be GPIO_PinSourcex where x can be (0..15).

Parameters

<i>GPIO_AFSelection</i>	<p>selects the pin to used as Alternate function. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • GPIO_AF_RTC_50Hz: Connect RTC_50Hz pin to AF0 (default after reset) • GPIO_AF_MCO: Connect MCO pin (MCO1 and MCO2) to AF0 (default after reset) • GPIO_AF_TAMPER: Connect TAMPER pins (TAMPER_1 and TAMPER_2) to AF0 (default after reset) • GPIO_AF_SWJ: Connect SWJ pins (SWD and JTAG)to AF0 (default after reset) • GPIO_AF_TRACE: Connect TRACE pins to AF0 (default after reset) • GPIO_AF_TIM1: Connect TIM1 pins to AF1 • GPIO_AF_TIM2: Connect TIM2 pins to AF1 • GPIO_AF_TIM3: Connect TIM3 pins to AF2 • GPIO_AF_TIM4: Connect TIM4 pins to AF2 • GPIO_AF_TIM5: Connect TIM5 pins to AF2 • GPIO_AF_TIM8: Connect TIM8 pins to AF3 • GPIO_AF_TIM9: Connect TIM9 pins to AF3 • GPIO_AF_TIM10: Connect TIM10 pins to AF3 • GPIO_AF_TIM11: Connect TIM11 pins to AF3 • GPIO_AF_I2C1: Connect I2C1 pins to AF4 • GPIO_AF_I2C2: Connect I2C2 pins to AF4 • GPIO_AF_I2C3: Connect I2C3 pins to AF4 • GPIO_AF_SPI1: Connect SPI1 pins to AF5 • GPIO_AF_SPI2: Connect SPI2/I2S2 pins to AF5 • GPIO_AF_SPI3: Connect SPI3/I2S3 pins to AF6 • GPIO_AF_I2S3ext: Connect I2S3ext pins to AF7 • GPIO_AF_USART1: Connect USART1 pins to AF7 • GPIO_AF_USART2: Connect USART2 pins to AF7 • GPIO_AF_USART3: Connect USART3 pins to AF7 • GPIO_AF_UART4: Connect UART4 pins to AF8 • GPIO_AF_UART5: Connect UART5 pins to AF8 • GPIO_AF_USART6: Connect USART6 pins to AF8 • GPIO_AF_CAN1: Connect CAN1 pins to AF9 • GPIO_AF_CAN2: Connect CAN2 pins to AF9 • GPIO_AF_TIM12: Connect TIM12 pins to AF9 • GPIO_AF_TIM13: Connect TIM13 pins to AF9 • GPIO_AF_TIM14: Connect TIM14 pins to AF9 • GPIO_AF_OTG_FS: Connect OTG_FS pins to AF10
Generated by Doxygen	<ul style="list-style-type: none"> • GPIO_AF_OTG_HS: Connect OTG_HS pins to AF10 • GPIO_AF_ETH: Connect ETHERNET pins to AF11 • GPIO_AF_FSMC: Connect FSMC pins to AF12

Parameters

Return values

None	
------	--

4.1.3.6 GPIO_Exported_Constants

Modules

- [GPIO_pins_define](#)
- [GPIO_Pin_sources](#)
- [GPIO_Alternat_function_selection_define](#)
- [GPIO_Legacy](#)

4.1.3.6.1 Detailed Description

4.1.3.6.2 GPIO_pins_define

Macros

- `#define GPIO_Pin_0 ((uint16_t)0x0001) /* Pin 0 selected */`
- `#define GPIO_Pin_1 ((uint16_t)0x0002) /* Pin 1 selected */`
- `#define GPIO_Pin_2 ((uint16_t)0x0004) /* Pin 2 selected */`
- `#define GPIO_Pin_3 ((uint16_t)0x0008) /* Pin 3 selected */`
- `#define GPIO_Pin_4 ((uint16_t)0x0010) /* Pin 4 selected */`
- `#define GPIO_Pin_5 ((uint16_t)0x0020) /* Pin 5 selected */`
- `#define GPIO_Pin_6 ((uint16_t)0x0040) /* Pin 6 selected */`
- `#define GPIO_Pin_7 ((uint16_t)0x0080) /* Pin 7 selected */`
- `#define GPIO_Pin_8 ((uint16_t)0x0100) /* Pin 8 selected */`
- `#define GPIO_Pin_9 ((uint16_t)0x0200) /* Pin 9 selected */`
- `#define GPIO_Pin_10 ((uint16_t)0x0400) /* Pin 10 selected */`
- `#define GPIO_Pin_11 ((uint16_t)0x0800) /* Pin 11 selected */`
- `#define GPIO_Pin_12 ((uint16_t)0x1000) /* Pin 12 selected */`
- `#define GPIO_Pin_13 ((uint16_t)0x2000) /* Pin 13 selected */`
- `#define GPIO_Pin_14 ((uint16_t)0x4000) /* Pin 14 selected */`
- `#define GPIO_Pin_15 ((uint16_t)0x8000) /* Pin 15 selected */`
- `#define GPIO_Pin_All ((uint16_t)0xFFFF) /* All pins selected */`
- `#define IS_GPIO_PIN(PIN) (((PIN) & (uint16_t)0x00) == 0x00) && ((PIN) != (uint16_t)0x00)`
- `#define IS_GET_GPIO_PIN(PIN)`

4.1.3.6.2.1 Detailed Description

4.1.3.6.2.2 Macro Definition Documentation

GPIO_Pin_0

```
#define GPIO_Pin_0 ((uint16_t)0x0001) /* Pin 0 selected */
```

GPIO_Pin_1

```
#define GPIO_Pin_1 ((uint16_t)0x0002) /* Pin 1 selected */
```

GPIO_Pin_10

```
#define GPIO_Pin_10 ((uint16_t)0x0400) /* Pin 10 selected */
```

GPIO_Pin_11

```
#define GPIO_Pin_11 ((uint16_t)0x0800) /* Pin 11 selected */
```

GPIO_Pin_12

```
#define GPIO_Pin_12 ((uint16_t)0x1000) /* Pin 12 selected */
```

GPIO_Pin_13

```
#define GPIO_Pin_13 ((uint16_t)0x2000) /* Pin 13 selected */
```

GPIO_Pin_14

```
#define GPIO_Pin_14 ((uint16_t)0x4000) /* Pin 14 selected */
```

GPIO_Pin_15

```
#define GPIO_Pin_15 ((uint16_t)0x8000) /* Pin 15 selected */
```

GPIO_Pin_2

```
#define GPIO_Pin_2 ((uint16_t)0x0004) /* Pin 2 selected */
```

GPIO_Pin_3

```
#define GPIO_Pin_3 ((uint16_t)0x0008) /* Pin 3 selected */
```

GPIO_Pin_4

```
#define GPIO_Pin_4 ((uint16_t)0x0010) /* Pin 4 selected */
```

GPIO_Pin_5

```
#define GPIO_Pin_5 ((uint16_t)0x0020) /* Pin 5 selected */
```

GPIO_Pin_6

```
#define GPIO_Pin_6 ((uint16_t)0x0040) /* Pin 6 selected */
```

GPIO_Pin_7

```
#define GPIO_Pin_7 ((uint16_t)0x0080) /* Pin 7 selected */
```

GPIO_Pin_8

```
#define GPIO_Pin_8 ((uint16_t)0x0100) /* Pin 8 selected */
```

GPIO_Pin_9

```
#define GPIO_Pin_9 ((uint16_t)0x0200) /* Pin 9 selected */
```

GPIO_Pin_All

```
#define GPIO_Pin_All ((uint16_t)0xFFFF) /* All pins selected */
```

IS_GET_GPIO_PIN

```
#define IS_GET_GPIO_PIN(  
    PIN )
```

Value:

```
((PIN) == GPIO_Pin_0) || \  
((PIN) == GPIO_Pin_1) || \  
((PIN) == GPIO_Pin_2) || \  
((PIN) == GPIO_Pin_3) || \  
((PIN) == GPIO_Pin_4) || \  
((PIN) == GPIO_Pin_5) || \  
((PIN) == GPIO_Pin_6) || \  
((PIN) == GPIO_Pin_7) || \  
((PIN) == GPIO_Pin_8) || \  
((PIN) == GPIO_Pin_9) || \  
((PIN) == GPIO_Pin_10) || \  
((PIN) == GPIO_Pin_11) || \  
((PIN) == GPIO_Pin_12) || \  
((PIN) == GPIO_Pin_13) || \  
((PIN) == GPIO_Pin_14) || \  
((PIN) == GPIO_Pin_15))
```

IS_GPIO_PIN

```
#define IS_GPIO_PIN(  
    PIN ) (((PIN) & (uint16_t)0x00) == 0x00) && ((PIN) != (uint16_t)0x00)
```

4.1.3.6.3 GPIO_Pin_sources

Macros

- `#define GPIO_PinSource0 ((uint8_t)0x00)`
- `#define GPIO_PinSource1 ((uint8_t)0x01)`
- `#define GPIO_PinSource2 ((uint8_t)0x02)`
- `#define GPIO_PinSource3 ((uint8_t)0x03)`
- `#define GPIO_PinSource4 ((uint8_t)0x04)`
- `#define GPIO_PinSource5 ((uint8_t)0x05)`
- `#define GPIO_PinSource6 ((uint8_t)0x06)`
- `#define GPIO_PinSource7 ((uint8_t)0x07)`
- `#define GPIO_PinSource8 ((uint8_t)0x08)`
- `#define GPIO_PinSource9 ((uint8_t)0x09)`
- `#define GPIO_PinSource10 ((uint8_t)0x0A)`
- `#define GPIO_PinSource11 ((uint8_t)0x0B)`
- `#define GPIO_PinSource12 ((uint8_t)0x0C)`
- `#define GPIO_PinSource13 ((uint8_t)0x0D)`
- `#define GPIO_PinSource14 ((uint8_t)0x0E)`
- `#define GPIO_PinSource15 ((uint8_t)0x0F)`
- `#define IS_GPIO_PIN_SOURCE(PINSOURCE)`

4.1.3.6.3.1 Detailed Description

4.1.3.6.3.2 Macro Definition Documentation

GPIO_PinSource0

```
#define GPIO_PinSource0 ((uint8_t)0x00)
```

GPIO_PinSource1

```
#define GPIO_PinSource1 ((uint8_t)0x01)
```

GPIO_PinSource10

```
#define GPIO_PinSource10 ((uint8_t)0x0A)
```

GPIO_PinSource11

```
#define GPIO_PinSource11 ((uint8_t)0x0B)
```

GPIO_PinSource12

```
#define GPIO_PinSource12 ((uint8_t)0x0C)
```

GPIO_PinSource13

```
#define GPIO_PinSource13 ((uint8_t)0x0D)
```

GPIO_PinSource14

```
#define GPIO_PinSource14 ((uint8_t)0x0E)
```

GPIO_PinSource15

```
#define GPIO_PinSource15 ((uint8_t)0x0F)
```

GPIO_PinSource2

```
#define GPIO_PinSource2 ((uint8_t)0x02)
```

GPIO_PinSource3

```
#define GPIO_PinSource3 ((uint8_t)0x03)
```

GPIO_PinSource4

```
#define GPIO_PinSource4 ((uint8_t)0x04)
```

GPIO_PinSource5

```
#define GPIO_PinSource5 ((uint8_t)0x05)
```

GPIO_PinSource6

```
#define GPIO_PinSource6 ((uint8_t)0x06)
```

GPIO_PinSource7

```
#define GPIO_PinSource7 ((uint8_t)0x07)
```

GPIO_PinSource8

```
#define GPIO_PinSource8 ((uint8_t)0x08)
```


GPIO_PinSource9

```
#define GPIO_PinSource9 ((uint8_t)0x09)
```

IS_GPIO_PIN_SOURCE

```
#define IS_GPIO_PIN_SOURCE(  
    PINSOURCE )
```

Value:

```
((PINSOURCE) == GPIO_PinSource0) || \  
((PINSOURCE) == GPIO_PinSource1) || \  
((PINSOURCE) == GPIO_PinSource2) || \  
((PINSOURCE) == GPIO_PinSource3) || \  
((PINSOURCE) == GPIO_PinSource4) || \  
((PINSOURCE) == GPIO_PinSource5) || \  
((PINSOURCE) == GPIO_PinSource6) || \  
((PINSOURCE) == GPIO_PinSource7) || \  
((PINSOURCE) == GPIO_PinSource8) || \  
((PINSOURCE) == GPIO_PinSource9) || \  
((PINSOURCE) == GPIO_PinSource10) || \  
((PINSOURCE) == GPIO_PinSource11) || \  
((PINSOURCE) == GPIO_PinSource12) || \  
((PINSOURCE) == GPIO_PinSource13) || \  
((PINSOURCE) == GPIO_PinSource14) || \  
((PINSOURCE) == GPIO_PinSource15)
```

4.1.3.6.4 GPIO_Alternat_function_selection_define**Macros**

- #define **GPIO_AF_RTC_50Hz** ((uint8_t)0x00) /* RTC_50Hz Alternate Function mapping */
 AF 0 selection
- #define **GPIO_AF_MCO** ((uint8_t)0x00) /* MCO (MCO1 and MCO2) Alternate Function mapping */
- #define **GPIO_AF_TAMPER** ((uint8_t)0x00) /* TAMPER (TAMPER_1 and TAMPER_2) Alternate Function mapping */
- #define **GPIO_AF_SWJ** ((uint8_t)0x00) /* SWJ (SWD and JTAG) Alternate Function mapping */
- #define **GPIO_AF_TRACE** ((uint8_t)0x00) /* TRACE Alternate Function mapping */
- #define **GPIO_AF_TIM1** ((uint8_t)0x01) /* TIM1 Alternate Function mapping */
 AF 1 selection
- #define **GPIO_AF_TIM2** ((uint8_t)0x01) /* TIM2 Alternate Function mapping */
- #define **GPIO_AF_TIM3** ((uint8_t)0x02) /* TIM3 Alternate Function mapping */
 AF 2 selection
- #define **GPIO_AF_TIM4** ((uint8_t)0x02) /* TIM4 Alternate Function mapping */
- #define **GPIO_AF_TIM5** ((uint8_t)0x02) /* TIM5 Alternate Function mapping */
- #define **GPIO_AF_TIM8** ((uint8_t)0x03) /* TIM8 Alternate Function mapping */
 AF 3 selection
- #define **GPIO_AF_TIM9** ((uint8_t)0x03) /* TIM9 Alternate Function mapping */
- #define **GPIO_AF_TIM10** ((uint8_t)0x03) /* TIM10 Alternate Function mapping */
- #define **GPIO_AF_TIM11** ((uint8_t)0x03) /* TIM11 Alternate Function mapping */
- #define **GPIO_AF_I2C1** ((uint8_t)0x04) /* I2C1 Alternate Function mapping */
 AF 4 selection
- #define **GPIO_AF_I2C2** ((uint8_t)0x04) /* I2C2 Alternate Function mapping */

- #define `GPIO_AF_I2C3` ((uint8_t)0x04) /* I2C3 Alternate Function mapping */
- #define `GPIO_AF_SPI1` ((uint8_t)0x05) /* SPI1 Alternate Function mapping */
- AF 5 selection*
- #define `GPIO_AF_SPI2` ((uint8_t)0x05) /* SPI2/I2S2 Alternate Function mapping */
- #define `GPIO_AF_SPI3` ((uint8_t)0x06) /* SPI3/I2S3 Alternate Function mapping */
- AF 6 selection*
- #define `GPIO_AF_USART1` ((uint8_t)0x07) /* USART1 Alternate Function mapping */
- AF 7 selection*
- #define `GPIO_AF_USART2` ((uint8_t)0x07) /* USART2 Alternate Function mapping */
- #define `GPIO_AF_USART3` ((uint8_t)0x07) /* USART3 Alternate Function mapping */
- #define `GPIO_AF_I2S3ext` ((uint8_t)0x07) /* I2S3ext Alternate Function mapping */
- #define `GPIO_AF_UART4` ((uint8_t)0x08) /* UART4 Alternate Function mapping */
- AF 8 selection*
- #define `GPIO_AF_UART5` ((uint8_t)0x08) /* UART5 Alternate Function mapping */
- #define `GPIO_AF_USART6` ((uint8_t)0x08) /* USART6 Alternate Function mapping */
- #define `GPIO_AF_CAN1` ((uint8_t)0x09) /* CAN1 Alternate Function mapping */
- AF 9 selection.*
- #define `GPIO_AF_CAN2` ((uint8_t)0x09) /* CAN2 Alternate Function mapping */
- #define `GPIO_AF_TIM12` ((uint8_t)0x09) /* TIM12 Alternate Function mapping */
- #define `GPIO_AF_TIM13` ((uint8_t)0x09) /* TIM13 Alternate Function mapping */
- #define `GPIO_AF_TIM14` ((uint8_t)0x09) /* TIM14 Alternate Function mapping */
- #define `GPIO_AF_OTG_FS` ((uint8_t)0xA) /* OTG_FS Alternate Function mapping */
- AF 10 selection*
- #define `GPIO_AF_OTG_HS` ((uint8_t)0xA) /* OTG_HS Alternate Function mapping */
- #define `GPIO_AF_ETH` ((uint8_t)0x0B) /* ETHERNET Alternate Function mapping */
- AF 11 selection*
- #define `GPIO_AF_FSMC` ((uint8_t)0xC) /* FSMC Alternate Function mapping */
- AF 12 selection*
- #define `GPIO_AF_OTG_HS_FS` ((uint8_t)0xC) /* OTG HS configured in FS, Alternate Function mapping */
- #define `GPIO_AF_SDIO` ((uint8_t)0xC) /* SDIO Alternate Function mapping */
- #define `GPIO_AF_DCM1` ((uint8_t)0x0D) /* DCM1 Alternate Function mapping */
- AF 13 selection*
- #define `GPIO_AF_EVENTOUT` ((uint8_t)0x0F) /* EVENTOUT Alternate Function mapping */
- AF 15 selection*
- #define `IS_GPIO_AF`(AF)

4.1.3.6.4.1 Detailed Description

4.1.3.6.4.2 Macro Definition Documentation

GPIO_AF_CAN1

```
#define GPIO_AF_CAN1 ((uint8_t)0x09) /* CAN1 Alternate Function mapping */
```

AF 9 selection.

GPIO_AF_CAN2

```
#define GPIO_AF_CAN2 ((uint8_t)0x09) /* CAN2 Alternate Function mapping */
```

GPIO_AF_DCMI

```
#define GPIO_AF_DCMI ((uint8_t)0x0D) /* DCMI Alternate Function mapping */
```

AF 13 selection

GPIO_AF_ETH

```
#define GPIO_AF_ETH ((uint8_t)0x0B) /* ETHERNET Alternate Function mapping */
```

AF 11 selection

GPIO_AF_EVENTOUT

```
#define GPIO_AF_EVENTOUT ((uint8_t)0x0F) /* EVENTOUT Alternate Function mapping */
```

AF 15 selection

GPIO_AF_FSMC

```
#define GPIO_AF_FSMC ((uint8_t)0xC) /* FSMC Alternate Function mapping */
```

AF 12 selection

GPIO_AF_I2C1

```
#define GPIO_AF_I2C1 ((uint8_t)0x04) /* I2C1 Alternate Function mapping */
```

AF 4 selection

GPIO_AF_I2C2

```
#define GPIO_AF_I2C2 ((uint8_t)0x04) /* I2C2 Alternate Function mapping */
```

GPIO_AF_I2C3

```
#define GPIO_AF_I2C3 ((uint8_t)0x04) /* I2C3 Alternate Function mapping */
```

GPIO_AF_I2S3ext

```
#define GPIO_AF_I2S3ext ((uint8_t)0x07) /* I2S3ext Alternate Function mapping */
```

GPIO_AF_MCO

```
#define GPIO_AF_MCO ((uint8_t)0x00) /* MCO (MCO1 and MCO2) Alternate Function mapping */
```

GPIO_AF_OTG_FS

```
#define GPIO_AF_OTG_FS ((uint8_t)0xA) /* OTG_FS Alternate Function mapping */
```

AF 10 selection

GPIO_AF_OTG_HS

```
#define GPIO_AF_OTG_HS ((uint8_t)0xA) /* OTG_HS Alternate Function mapping */
```

GPIO_AF_OTG_HS_FS

```
#define GPIO_AF_OTG_HS_FS ((uint8_t)0xC) /* OTG HS configured in FS, Alternate Function mapping */
```

GPIO_AF_RTC_50Hz

```
#define GPIO_AF_RTC_50Hz ((uint8_t)0x00) /* RTC_50Hz Alternate Function mapping */
```

AF 0 selection

GPIO_AF_SDIO

```
#define GPIO_AF_SDIO ((uint8_t)0xC) /* SDIO Alternate Function mapping */
```

GPIO_AF_SPI1

```
#define GPIO_AF_SPI1 ((uint8_t)0x05) /* SPI1 Alternate Function mapping */
```

AF 5 selection

GPIO_AF_SPI2

```
#define GPIO_AF_SPI2 ((uint8_t)0x05) /* SPI2/I2S2 Alternate Function mapping */
```

GPIO_AF_SPI3

```
#define GPIO_AF_SPI3 ((uint8_t)0x06) /* SPI3/I2S3 Alternate Function mapping */
```

AF 6 selection

GPIO_AF_SWJ

```
#define GPIO_AF_SWJ ((uint8_t)0x00) /* SWJ (SWD and JTAG) Alternate Function mapping */
```

GPIO_AF_TAMPER

```
#define GPIO_AF_TAMPER ((uint8_t)0x00) /* TAMPER (TAMPER_1 and TAMPER_2) Alternate Function mapping */
```

GPIO_AF_TIM1

```
#define GPIO_AF_TIM1 ((uint8_t)0x01) /* TIM1 Alternate Function mapping */
```

AF 1 selection

GPIO_AF_TIM10

```
#define GPIO_AF_TIM10 ((uint8_t)0x03) /* TIM10 Alternate Function mapping */
```

GPIO_AF_TIM11

```
#define GPIO_AF_TIM11 ((uint8_t)0x03) /* TIM11 Alternate Function mapping */
```

GPIO_AF_TIM12

```
#define GPIO_AF_TIM12 ((uint8_t)0x09) /* TIM12 Alternate Function mapping */
```

GPIO_AF_TIM13

```
#define GPIO_AF_TIM13 ((uint8_t)0x09) /* TIM13 Alternate Function mapping */
```

GPIO_AF_TIM14

```
#define GPIO_AF_TIM14 ((uint8_t)0x09) /* TIM14 Alternate Function mapping */
```

GPIO_AF_TIM2

```
#define GPIO_AF_TIM2 ((uint8_t)0x01) /* TIM2 Alternate Function mapping */
```

GPIO_AF_TIM3

```
#define GPIO_AF_TIM3 ((uint8_t)0x02) /* TIM3 Alternate Function mapping */
```

AF 2 selection

GPIO_AF_TIM4

```
#define GPIO_AF_TIM4 ((uint8_t)0x02) /* TIM4 Alternate Function mapping */
```

GPIO_AF_TIM5

```
#define GPIO_AF_TIM5 ((uint8_t)0x02) /* TIM5 Alternate Function mapping */
```

GPIO_AF_TIM8

```
#define GPIO_AF_TIM8 ((uint8_t)0x03) /* TIM8 Alternate Function mapping */
```

AF 3 selection

GPIO_AF_TIM9

```
#define GPIO_AF_TIM9 ((uint8_t)0x03) /* TIM9 Alternate Function mapping */
```

GPIO_AF_TRACE

```
#define GPIO_AF_TRACE ((uint8_t)0x00) /* TRACE Alternate Function mapping */
```

GPIO_AF_UART4

```
#define GPIO_AF_UART4 ((uint8_t)0x08) /* UART4 Alternate Function mapping */
```

AF 8 selection

GPIO_AF_UART5

```
#define GPIO_AF_UART5 ((uint8_t)0x08) /* UART5 Alternate Function mapping */
```

GPIO_AF_USART1

```
#define GPIO_AF_USART1 ((uint8_t)0x07) /* USART1 Alternate Function mapping */
```

AF 7 selection

GPIO_AF_USART2

```
#define GPIO_AF_USART2 ((uint8_t)0x07) /* USART2 Alternate Function mapping */
```

GPIO_AF_USART3

```
#define GPIO_AF_USART3 ((uint8_t)0x07) /* USART3 Alternate Function mapping */
```

GPIO_AF_USART6

```
#define GPIO_AF_USART6 ((uint8_t)0x08) /* USART6 Alternate Function mapping */
```

IS_GPIO_AF

```
#define IS_GPIO_AF(  
    AF )
```

Value:

```
((AF) == GPIO_AF_RTC_50Hz) || ((AF) == GPIO_AF_TIM14) || \
((AF) == GPIO_AF_MCO)      || ((AF) == GPIO_AF_TAMPER) || \
((AF) == GPIO_AF_SWJ)      || ((AF) == GPIO_AF_TRACE)  || \
((AF) == GPIO_AF_TIM1)     || ((AF) == GPIO_AF_TIM2)   || \
((AF) == GPIO_AF_TIM3)     || ((AF) == GPIO_AF_TIM4)   || \
((AF) == GPIO_AF_TIM5)     || ((AF) == GPIO_AF_TIM8)   || \
((AF) == GPIO_AF_I2C1)     || ((AF) == GPIO_AF_I2C2)   || \
((AF) == GPIO_AF_I2C3)     || ((AF) == GPIO_AF_SPI1)   || \
((AF) == GPIO_AF_SPI2)     || ((AF) == GPIO_AF_TIM13)  || \
((AF) == GPIO_AF_SPI3)     || ((AF) == GPIO_AF_TIM14)  || \
((AF) == GPIO_AF_USART1)   || ((AF) == GPIO_AF_USART2) || \
((AF) == GPIO_AF_USART3)   || ((AF) == GPIO_AF_UART4)  || \
((AF) == GPIO_AF_UART5)    || ((AF) == GPIO_AF_USART6) || \
((AF) == GPIO_AF_CAN1)     || ((AF) == GPIO_AF_CAN2)   || \
((AF) == GPIO_AF_OTG_FS)   || ((AF) == GPIO_AF_OTG_HS) || \
((AF) == GPIO_AF_ETH)      || ((AF) == GPIO_AF_FSMC)   || \
((AF) == GPIO_AF_OTG_HS_FS) || ((AF) == GPIO_AF_SDIO)  || \
((AF) == GPIO_AF_DCMI)     || ((AF) == GPIO_AF_EVENTOUT)
```

4.1.3.6.5 GPIO_Legacy**Macros**

- #define [GPIO_Mode_AIN](#) [GPIO_Mode_AN](#)
- #define [GPIO_AF_OTG1_FS](#) [GPIO_AF_OTG_FS](#)
- #define [GPIO_AF_OTG2_HS](#) [GPIO_AF_OTG_HS](#)
- #define [GPIO_AF_OTG2_FS](#) [GPIO_AF_OTG_HS_FS](#)

4.1.3.6.5.1 Detailed Description

4.1.3.6.5.2 Macro Definition Documentation

GPIO_AF_OTG1_FS

```
#define GPIO_AF_OTG1_FS GPIO\_AF\_OTG\_FS
```

GPIO_AF_OTG2_FS

```
#define GPIO_AF_OTG2_FS GPIO\_AF\_OTG\_HS\_FS
```

GPIO_AF_OTG2_HS

```
#define GPIO_AF_OTG2_HS GPIO\_AF\_OTG\_HS
```

GPIO_Mode_AIN

```
#define GPIO_Mode_AIN GPIO\_Mode\_AN
```

4.1.4 I2C

I2C driver modules.

Modules

- [I2C_Private_Functions](#)
- [I2C_Exported_Constants](#)

Data Structures

- struct [I2C_InitTypeDef](#)
I2C Init structure definition

Macros

- #define [CR1_CLEAR_MASK](#) ((uint16_t)0xFBF5) /*<! I2C registers Masks */
- #define [FLAG_MASK](#) ((uint32_t)0x0FFFFFFF) /*<! I2C FLAG mask */
- #define [ITEN_MASK](#) ((uint32_t)0x07000000) /*<! I2C Interrupt Enable mask */

Functions

- void **I2C_DeInit** (I2C_TypeDef *I2Cx)
Deinitialize the I2Cx peripheral registers to their default reset values.
- void **I2C_Init** (I2C_TypeDef *I2Cx, I2C_InitTypeDef *I2C_InitStruct)
Initializes the I2Cx peripheral according to the specified parameters in the I2C_InitStruct.
- void **I2C_StructInit** (I2C_InitTypeDef *I2C_InitStruct)
Fills each I2C_InitStruct member with its default value.
- void **I2C_Cmd** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C peripheral.
- void **I2C_GenerateSTART** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Generates I2Cx communication START condition.
- void **I2C_GenerateSTOP** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Generates I2Cx communication STOP condition.
- void **I2C_Send7bitAddress** (I2C_TypeDef *I2Cx, uint8_t Address, uint8_t I2C_Direction)
Transmits the address byte to select the slave device.
- void **I2C_AcknowledgeConfig** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C acknowledge feature.
- void **I2C_OwnAddress2Config** (I2C_TypeDef *I2Cx, uint8_t Address)
Configures the specified I2C own address2.
- void **I2C_DualAddressCmd** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C dual addressing mode.
- void **I2C_GeneralCallCmd** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C general call feature.
- void **I2C_SoftwareResetCmd** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C software reset.
- void **I2C_StretchClockCmd** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C Clock stretching.
- void **I2C_FastModeDutyCycleConfig** (I2C_TypeDef *I2Cx, uint16_t I2C_DutyCycle)
Selects the specified I2C fast mode duty cycle.
- void **I2C_NACKPositionConfig** (I2C_TypeDef *I2Cx, uint16_t I2C_NACKPosition)
Selects the specified I2C NACK position in master receiver mode.
- void **I2C_SMBusAlertConfig** (I2C_TypeDef *I2Cx, uint16_t I2C_SMBusAlert)
Drives the SMBusAlert pin high or low for the specified I2C.
- void **I2C_ARPCmd** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C ARP.
- void **I2C_SendData** (I2C_TypeDef *I2Cx, uint8_t Data)
Sends a data byte through the I2Cx peripheral.
- uint8_t **I2C_ReceiveData** (I2C_TypeDef *I2Cx)
Returns the most recent received data by the I2Cx peripheral.
- void **I2C_TransmitPEC** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C PEC transfer.
- void **I2C_PECPositionConfig** (I2C_TypeDef *I2Cx, uint16_t I2C_PECPosition)
Selects the specified I2C PEC position.
- void **I2C_CalculatePEC** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the PEC value calculation of the transferred bytes.
- uint8_t **I2C_GetPEC** (I2C_TypeDef *I2Cx)
Returns the PEC value for the specified I2C.
- void **I2C_DMAMCmd** (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C DMA requests.
- void **I2C_DMALastTransferCmd** (I2C_TypeDef *I2Cx, FunctionalState NewState)

Specifies that the next DMA transfer is the last one.

- uint16_t [I2C_ReadRegister](#) (I2C_TypeDef *I2Cx, uint8_t I2C_Register)
Reads the specified I2C register and returns its value.
- void [I2C_ITConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_IT, FunctionalState NewState)
Enables or disables the specified I2C interrupts.
- ErrorStatus [I2C_CheckEvent](#) (I2C_TypeDef *I2Cx, uint32_t I2C_EVENT)
Checks whether the last I2Cx Event is equal to the one passed as parameter.
- uint32_t [I2C_GetLastEvent](#) (I2C_TypeDef *I2Cx)
Returns the last I2Cx Event.
- FlagStatus [I2C_GetFlagStatus](#) (I2C_TypeDef *I2Cx, uint32_t I2C_FLAG)
Checks whether the specified I2C flag is set or not.
- void [I2C_ClearFlag](#) (I2C_TypeDef *I2Cx, uint32_t I2C_FLAG)
Clears the I2Cx's pending flags.
- ITStatus [I2C_GetITStatus](#) (I2C_TypeDef *I2Cx, uint32_t I2C_IT)
Checks whether the specified I2C interrupt has occurred or not.
- void [I2C_ClearITPendingBit](#) (I2C_TypeDef *I2Cx, uint32_t I2C_IT)
Clears the I2Cx's interrupt pending bits.

4.1.4.1 Detailed Description

I2C driver modules.

4.1.4.2 Macro Definition Documentation

4.1.4.2.1 CR1_CLEAR_MASK

```
#define CR1_CLEAR_MASK ((uint16_t)0xFBF5) /*<! I2C registers Masks */
```

4.1.4.2.2 FLAG_MASK

```
#define FLAG_MASK ((uint32_t)0x00FFFFFF) /*<! I2C FLAG mask */
```

4.1.4.2.3 ITEN_MASK

```
#define ITEN_MASK ((uint32_t)0x07000000) /*<! I2C Interrupt Enable mask */
```

4.1.4.3 Function Documentation

4.1.4.3.1 I2C_AcknowledgeConfig()

```
void I2C_AcknowledgeConfig (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C acknowledge feature.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C Acknowledgement. This parameter can be: ENABLE or DISABLE.

Return values

<i>None.</i>	
--------------	--

4.1.4.3.2 I2C_ARPCmd()

```
void I2C_ARPCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C ARP.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2Cx ARP. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.3.3 I2C_CalculatePEC()

```
void I2C_CalculatePEC (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the PEC value calculation of the transferred bytes.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2Cx PEC value calculation. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.3.4 I2C_CheckEvent()

```
ErrorStatus I2C_CheckEvent (
```

```
I2C_TypeDef * I2Cx,
uint32_t I2C_EVENT )
```

Checks whether the last I2Cx Event is equal to the one passed as parameter.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_EVENT</i>	<p>specifies the event to be checked. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED: EV1 • I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED: EV1 • I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED: EV1 • I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED: EV1 • I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED: EV1 • I2C_EVENT_SLAVE_BYTE_RECEIVED: EV2 • (I2C_EVENT_SLAVE_BYTE_RECEIVED I2C_FLAG_DUALF): EV2 • (I2C_EVENT_SLAVE_BYTE_RECEIVED I2C_FLAG_GENCALL): EV2 • I2C_EVENT_SLAVE_BYTE_TRANSMITTED: EV3 • (I2C_EVENT_SLAVE_BYTE_TRANSMITTED I2C_FLAG_DUALF): EV3 • (I2C_EVENT_SLAVE_BYTE_TRANSMITTED I2C_FLAG_GENCALL): EV3 • I2C_EVENT_SLAVE_ACK_FAILURE: EV3_2 • I2C_EVENT_SLAVE_STOP_DETECTED: EV4 • I2C_EVENT_MASTER_MODE_SELECT: EV5 • I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED: EV6 • I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED: EV6 • I2C_EVENT_MASTER_BYTE_RECEIVED: EV7 • I2C_EVENT_MASTER_BYTE_TRANSMITTING: EV8 • I2C_EVENT_MASTER_BYTE_TRANSMITTED: EV8_2 • I2C_EVENT_MASTER_MODE_ADDRESS10: EV9

Note

For detailed description of Events, please refer to section I2C_Events in [stm32f4xx_i2c.h](#) file.

Return values

<i>An</i>	<p>ErrorStatus enumeration value:</p> <ul style="list-style-type: none"> • SUCCESS: Last event is equal to the I2C_EVENT • ERROR: Last event is different from the I2C_EVENT
-----------	--

4.1.4.3.5 I2C_ClearFlag()

```
void I2C_ClearFlag (
    I2C_TypeDef * I2Cx,
    uint32_t I2C_FLAG )
```

Clears the I2Cx's pending flags.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_FLAG</i>	specifies the flag to clear. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • I2C_FLAG_SMBALERT: SMBus Alert flag • I2C_FLAG_TIMEOUT: Timeout or Tlow error flag • I2C_FLAG_PECERR: PEC error in reception flag • I2C_FLAG_OVR: Overrun/Underrun flag (Slave mode) • I2C_FLAG_AF: Acknowledge failure flag • I2C_FLAG_ARLO: Arbitration lost flag (Master mode) • I2C_FLAG_BERR: Bus error flag

Note

STOPF (STOP detection) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetFlagStatus\(\)](#)) followed by a write operation to I2C_CR1 register ([I2C_Cmd\(\)](#)) to re-enable the I2C peripheral).

ADD10 (10-bit header sent) is cleared by software sequence: a read operation to I2C_SR1 ([I2C_GetFlagStatus\(\)](#)) followed by writing the second byte of the address in DR register.

BTF (Byte Transfer Finished) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetFlagStatus\(\)](#)) followed by a read/write to I2C_DR register ([I2C_SendData\(\)](#)).

ADDR (Address sent) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetFlagStatus\(\)](#)) followed by a read operation to I2C_SR2 register ((void)(I2Cx->SR2)).

SB (Start Bit) is cleared software sequence: a read operation to I2C_SR1 register ([I2C_GetFlagStatus\(\)](#)) followed by a write operation to I2C_DR register ([I2C_SendData\(\)](#)).

Return values

<i>None</i>	
-------------	--

4.1.4.3.6 I2C_ClearITPendingBit()

```
void I2C_ClearITPendingBit (
    I2C_TypeDef * I2Cx,
    uint32_t I2C_IT )
```

Clears the I2Cx's interrupt pending bits.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C↔ _IT</i>	<p>specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • I2C_IT_SMBALERT: SMBus Alert interrupt • I2C_IT_TIMEOUT: Timeout or Tlow error interrupt • I2C_IT_PECERR: PEC error in reception interrupt • I2C_IT_OVR: Overrun/Underrun interrupt (Slave mode) • I2C_IT_AF: Acknowledge failure interrupt • I2C_IT_ARLO: Arbitration lost interrupt (Master mode) • I2C_IT_BERR: Bus error interrupt

Note

STOPF (STOP detection) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetITStatus\(\)](#)) followed by a write operation to I2C_CR1 register ([I2C_Cmd\(\)](#)) to re-enable the I2C peripheral).

ADD10 (10-bit header sent) is cleared by software sequence: a read operation to I2C_SR1 ([I2C_GetITStatus\(\)](#)) followed by writing the second byte of the address in I2C_DR register.

BTF (Byte Transfer Finished) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetITStatus\(\)](#)) followed by a read/write to I2C_DR register ([I2C_SendData\(\)](#)).

ADDR (Address sent) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetITStatus\(\)](#)) followed by a read operation to I2C_SR2 register ((void)(I2Cx->SR2)).

SB (Start Bit) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetITStatus\(\)](#)) followed by a write operation to I2C_DR register ([I2C_SendData\(\)](#)).

Return values

<i>None</i>	
-------------	--

4.1.4.3.7 I2C_Cmd()

```
void I2C_Cmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C peripheral.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2Cx peripheral. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.3.8 I2C_DeInit()

```
void I2C_DeInit (
    I2C_TypeDef * I2Cx )
```

Deinitialize the I2Cx peripheral registers to their default reset values.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
-------------	--

Return values

<i>None</i>	
-------------	--

4.1.4.3.9 I2C_DMAMCmd()

```
void I2C_DMAMCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C DMA requests.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C DMA transfer. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.3.10 I2C_DMALastTransferCmd()

```
void I2C_DMALastTransferCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Specifies that the next DMA transfer is the last one.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C DMA last transfer. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.3.11 I2C_DualAddressCmd()

```
void I2C_DualAddressCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C dual addressing mode.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C dual addressing mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.3.12 I2C_FastModeDutyCycleConfig()

```
void I2C_FastModeDutyCycleConfig (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_DutyCycle )
```

Selects the specified I2C fast mode duty cycle.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_DutyCycle</i>	specifies the fast mode duty cycle. This parameter can be one of the following values: <ul style="list-style-type: none"> I2C_DutyCycle_2: I2C fast mode Tlow/Thigh = 2 I2C_DutyCycle_16_9: I2C fast mode Tlow/Thigh = 16/9

Return values

<i>None</i>	
-------------	--

4.1.4.3.13 I2C_GeneralCallCmd()

```
void I2C_GeneralCallCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```


Enables or disables the specified I2C general call feature.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C General call. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.3.14 I2C_GenerateSTART()

```
void I2C_GenerateSTART (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Generates I2Cx communication START condition.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C START condition generation. This parameter can be: ENABLE or DISABLE.

Return values

<i>None.</i>	
--------------	--

4.1.4.3.15 I2C_GenerateSTOP()

```
void I2C_GenerateSTOP (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Generates I2Cx communication STOP condition.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C STOP condition generation. This parameter can be: ENABLE or DISABLE.

Return values

<i>None.</i>	
--------------	--

4.1.4.3.16 I2C_GetFlagStatus()

```
FlagStatus I2C_GetFlagStatus (
    I2C_TypeDef * I2Cx,
    uint32_t I2C_FLAG )
```

Checks whether the specified I2C flag is set or not.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_FLAG</i>	<p>specifies the flag to check. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • I2C_FLAG_DUALF: Dual flag (Slave mode) • I2C_FLAG_SMBHOST: SMBus host header (Slave mode) • I2C_FLAG_SMBDEFAULT: SMBus default header (Slave mode) • I2C_FLAG_GENCALL: General call header flag (Slave mode) • I2C_FLAG_TRA: Transmitter/Receiver flag • I2C_FLAG_BUSY: Bus busy flag • I2C_FLAG_MSL: Master/Slave flag • I2C_FLAG_SMBALERT: SMBus Alert flag • I2C_FLAG_TIMEOUT: Timeout or Tlow error flag • I2C_FLAG_PECERR: PEC error in reception flag • I2C_FLAG_OVR: Overrun/Underrun flag (Slave mode) • I2C_FLAG_AF: Acknowledge failure flag • I2C_FLAG_ARLO: Arbitration lost flag (Master mode) • I2C_FLAG_BERR: Bus error flag • I2C_FLAG_TXE: Data register empty flag (Transmitter) • I2C_FLAG_RXNE: Data register not empty (Receiver) flag • I2C_FLAG_STOPF: Stop detection flag (Slave mode) • I2C_FLAG_ADD10: 10-bit header sent flag (Master mode) • I2C_FLAG_BTF: Byte transfer finished flag • I2C_FLAG_ADDR: Address sent flag (Master mode) "ADSL" Address matched flag (Slave mode)"ENDAD" • I2C_FLAG_SB: Start bit flag (Master mode)

Return values

<i>The</i>	new state of I2C_FLAG (SET or RESET).
------------	---------------------------------------

4.1.4.3.17 I2C_GetITStatus()

```
ITStatus I2C_GetITStatus (
    I2C_TypeDef * I2Cx,
    uint32_t I2C_IT )
```

Checks whether the specified I2C interrupt has occurred or not.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_ _IT</i>	<p>specifies the interrupt source to check. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • I2C_IT_SMBALERT: SMBus Alert flag • I2C_IT_TIMEOUT: Timeout or Tlow error flag • I2C_IT_PECERR: PEC error in reception flag • I2C_IT_OVR: Overrun/Underrun flag (Slave mode) • I2C_IT_AF: Acknowledge failure flag • I2C_IT_ARLO: Arbitration lost flag (Master mode) • I2C_IT_BERR: Bus error flag • I2C_IT_TXE: Data register empty flag (Transmitter) • I2C_IT_RXNE: Data register not empty (Receiver) flag • I2C_IT_STOPF: Stop detection flag (Slave mode) • I2C_IT_ADD10: 10-bit header sent flag (Master mode) • I2C_IT_BTF: Byte transfer finished flag • I2C_IT_ADDR: Address sent flag (Master mode) "ADSL" Address matched flag (Slave mode)"ENDAD" • I2C_IT_SB: Start bit flag (Master mode)

Return values

<i>The</i>	new state of I2C_IT (SET or RESET).
------------	-------------------------------------

4.1.4.3.18 I2C_GetLastEvent()

```
uint32_t I2C_GetLastEvent (
    I2C_TypeDef * I2Cx )
```

Returns the last I2Cx Event.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
-------------	--

Note

For detailed description of Events, please refer to section I2C_Events in [stm32f4xx_i2c.h](#) file.

Return values

<i>The</i>	last event
------------	------------

4.1.4.3.19 I2C_GetPEC()

```
uint8_t I2C_GetPEC (
    I2C_TypeDef * I2Cx )
```

Returns the PEC value for the specified I2C.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
-------------	--

Return values

<i>The</i>	PEC value.
------------	------------

4.1.4.3.20 I2C_Init()

```
void I2C_Init (
    I2C_TypeDef * I2Cx,
    I2C_InitTypeDef * I2C_InitStruct )
```

Initializes the I2Cx peripheral according to the specified parameters in the I2C_InitStruct.

Note

To use the I2C at 400 KHz (in fast mode), the PCLK1 frequency (I2C peripheral input clock) must be a multiple of 10 MHz.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_InitStruct</i>	pointer to a I2C_InitTypeDef structure that contains the configuration information for the specified I2C peripheral.

Return values

<i>None</i>	
-------------	--

4.1.4.3.21 I2C_ITConfig()

```
void I2C_ITConfig (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_IT,
    FunctionalState NewState )
```

Enables or disables the specified I2C interrupts.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_IT</i>	specifies the I2C interrupts sources to be enabled or disabled. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • I2C_IT_BUF: Buffer interrupt mask • I2C_IT_EVT: Event interrupt mask • I2C_IT_ERR: Error interrupt mask
<i>NewState</i>	new state of the specified I2C interrupts. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.3.22 I2C_NACKPositionConfig()

```
void I2C_NACKPositionConfig (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_NACKPosition )
```

Selects the specified I2C NACK position in master receiver mode.

Note

This function is useful in I2C Master Receiver mode when the number of data to be received is equal to 2. In this case, this function should be called (with parameter I2C_NACKPosition_Next) before data reception starts, as described in the 2-byte reception procedure recommended in Reference Manual in Section: Master receiver.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_NACKPosition</i>	specifies the NACK position. This parameter can be one of the following values: <ul style="list-style-type: none"> • I2C_NACKPosition_Next: indicates that the next byte will be the last received byte. • I2C_NACKPosition_Current: indicates that current byte is the last received byte.

Note

This function configures the same bit (POS) as [I2C_PECPositionConfig\(\)](#) but is intended to be used in I2C mode while [I2C_PECPositionConfig\(\)](#) is intended to be used in SMBUS mode.

Return values

<i>None</i>	
-------------	--

4.1.4.3.23 I2C_OwnAddress2Config()

```
void I2C_OwnAddress2Config (
    I2C_TypeDef * I2Cx,
    uint8_t Address )
```

Configures the specified I2C own address2.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>Address</i>	specifies the 7bit I2C own address2.

Return values

<i>None.</i>	
--------------	--

4.1.4.3.24 I2C_PECPositionConfig()

```
void I2C_PECPositionConfig (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_PECPosition )
```

Selects the specified I2C PEC position.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_PECPosition</i>	specifies the PEC position. This parameter can be one of the following values: <ul style="list-style-type: none"> I2C_PECPosition_Next: indicates that the next byte is PEC I2C_PECPosition_Current: indicates that current byte is PEC

Note

This function configures the same bit (POS) as [I2C_NACKPositionConfig\(\)](#) but is intended to be used in SMBUS mode while [I2C_NACKPositionConfig\(\)](#) is intended to be used in I2C mode.

Return values

<i>None</i>	
-------------	--

4.1.4.3.25 I2C_ReadRegister()

```
uint16_t I2C_ReadRegister (
    I2C_TypeDef * I2Cx,
    uint8_t I2C_Register )
```

Reads the specified I2C register and returns its value.

Parameters

<i>I2C_Register</i>	<p>specifies the register to read. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • I2C_Register_CR1: CR1 register. • I2C_Register_CR2: CR2 register. • I2C_Register_OAR1: OAR1 register. • I2C_Register_OAR2: OAR2 register. • I2C_Register_DR: DR register. • I2C_Register_SR1: SR1 register. • I2C_Register_SR2: SR2 register. • I2C_Register_CCR: CCR register. • I2C_Register_TRISE: TRISE register.
---------------------	---

Return values

<i>The</i>	value of the read register.
------------	-----------------------------

4.1.4.3.26 I2C_ReceiveData()

```
uint8_t I2C_ReceiveData (
    I2C_TypeDef * I2Cx )
```

Returns the most recent received data by the I2Cx peripheral.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
-------------	--

Return values

<i>The</i>	value of the received data.
------------	-----------------------------

4.1.4.3.27 I2C_Send7bitAddress()

```
void I2C_Send7bitAddress (
    I2C_TypeDef * I2Cx,
    uint8_t Address,
    uint8_t I2C_Direction )
```

Transmits the address byte to select the slave device.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>Address</i>	specifies the slave address which will be transmitted
<i>I2C_Direction</i>	specifies whether the I2C device will be a Transmitter or a Receiver. This parameter can be one of the following values <ul style="list-style-type: none"> I2C_Direction_Transmitter: Transmitter mode I2C_Direction_Receiver: Receiver mode

Return values

<i>None.</i>	
--------------	--

4.1.4.3.28 I2C_SendData()

```
void I2C_SendData (
    I2C_TypeDef * I2Cx,
    uint8_t Data )
```

Sends a data byte through the I2Cx peripheral.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>Data</i>	Byte to be transmitted..

Return values

<i>None</i>	
-------------	--

4.1.4.3.29 I2C_SMBusAlertConfig()

```
void I2C_SMBusAlertConfig (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_SMBusAlert )
```

Drives the SMBusAlert pin high or low for the specified I2C.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_SMBusAlert</i>	specifies SMBAlert pin level. This parameter can be one of the following values: <ul style="list-style-type: none"> • I2C_SMBusAlert_Low: SMBAlert pin driven low • I2C_SMBusAlert_High: SMBAlert pin driven high

Return values

<i>None</i>	
-------------	--

4.1.4.3.30 I2C_SoftwareResetCmd()

```
void I2C_SoftwareResetCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C software reset.

Note

When software reset is enabled, the I2C IOs are released (this can be useful to recover from bus errors).

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C software reset. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.3.31 I2C_StretchClockCmd()

```
void I2C_StretchClockCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C Clock stretching.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2Cx Clock stretching. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.3.32 I2C_StructInit()

```
void I2C_StructInit (
    I2C_InitTypeDef * I2C_InitStruct )
```

Fills each I2C_InitStruct member with its default value.

Parameters

<i>I2C_InitStruct</i>	pointer to an I2C_InitTypeDef structure which will be initialized.
-----------------------	--

Return values

<i>None</i>	
-------------	--

4.1.4.3.33 I2C_TransmitPEC()

```
void I2C_TransmitPEC (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C PEC transfer.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C PEC transmission. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.4 I2C_Private_Functions**Modules**

- [Initialization and Configuration functions](#)
Initialization and Configuration functions.
- [Data transfers functions](#)
Data transfers functions.
- [PEC management functions](#)

PEC management functions.

- [DMA transfers management functions](#)

DMA transfers management functions.

- [Interrupts events and flags management functions](#)

Interrupts, events and flags management functions.

4.1.4.4.1 Detailed Description

4.1.4.4.2 Initialization and Configuration functions

Initialization and Configuration functions.

Functions

- void [I2C_DeInit](#) (I2C_TypeDef *I2Cx)
Deinitialize the I2Cx peripheral registers to their default reset values.
- void [I2C_Init](#) (I2C_TypeDef *I2Cx, [I2C_InitTypeDef](#) *I2C_InitStruct)
Initializes the I2Cx peripheral according to the specified parameters in the I2C_InitStruct.
- void [I2C_StructInit](#) ([I2C_InitTypeDef](#) *I2C_InitStruct)
Fills each I2C_InitStruct member with its default value.
- void [I2C_Cmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C peripheral.
- void [I2C_GenerateSTART](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Generates I2Cx communication START condition.
- void [I2C_GenerateSTOP](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Generates I2Cx communication STOP condition.
- void [I2C_Send7bitAddress](#) (I2C_TypeDef *I2Cx, uint8_t Address, uint8_t I2C_Direction)
Transmits the address byte to select the slave device.
- void [I2C_AcknowledgeConfig](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C acknowledge feature.
- void [I2C_OwnAddress2Config](#) (I2C_TypeDef *I2Cx, uint8_t Address)
Configures the specified I2C own address2.
- void [I2C_DualAddressCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C dual addressing mode.
- void [I2C_GeneralCallCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C general call feature.
- void [I2C_SoftwareResetCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C software reset.
- void [I2C_StretchClockCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C Clock stretching.
- void [I2C_FastModeDutyCycleConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_DutyCycle)
Selects the specified I2C fast mode duty cycle.
- void [I2C_NACKPositionConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_NACKPosition)
Selects the specified I2C NACK position in master receiver mode.
- void [I2C_SMBusAlertConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_SMBusAlert)
Drives the SMBusAlert pin high or low for the specified I2C.
- void [I2C_ARPCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C ARP.

4.1.4.4.2.1 Detailed Description

Initialization and Configuration functions.

```
=====
Initialization and Configuration functions
=====
```

4.1.4.4.2.2 Function Documentation

I2C_AcknowledgeConfig()

```
void I2C_AcknowledgeConfig (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C acknowledge feature.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C Acknowledgement. This parameter can be: ENABLE or DISABLE.

Return values

<i>None.</i>	
--------------	--

I2C_ARPCmd()

```
void I2C_ARPCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C ARP.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2Cx ARP. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

I2C_Cmd()

```
void I2C_Cmd (
```

```
I2C_TypeDef * I2Cx,  
FunctionalState NewState )
```

Enables or disables the specified I2C peripheral.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2Cx peripheral. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

I2C_DeInit()

```
void I2C_DeInit (  
    I2C_TypeDef * I2Cx )
```

Deinitialize the I2Cx peripheral registers to their default reset values.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
-------------	--

Return values

<i>None</i>	
-------------	--

I2C_DualAddressCmd()

```
void I2C_DualAddressCmd (  
    I2C_TypeDef * I2Cx,  
    FunctionalState NewState )
```

Enables or disables the specified I2C dual addressing mode.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C dual addressing mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

I2C_FastModeDutyCycleConfig()

```
void I2C_FastModeDutyCycleConfig (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_DutyCycle )
```

Selects the specified I2C fast mode duty cycle.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_DutyCycle</i>	specifies the fast mode duty cycle. This parameter can be one of the following values: <ul style="list-style-type: none"> • I2C_DutyCycle_2: I2C fast mode Tlow/Thigh = 2 • I2C_DutyCycle_16_9: I2C fast mode Tlow/Thigh = 16/9

Return values

<i>None</i>	
-------------	--

I2C_GeneralCallCmd()

```
void I2C_GeneralCallCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C general call feature.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C General call. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

I2C_GenerateSTART()

```
void I2C_GenerateSTART (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Generates I2Cx communication START condition.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C START condition generation. This parameter can be: ENABLE or DISABLE.

Return values

None.	
-------	--

I2C_GenerateSTOP()

```
void I2C_GenerateSTOP (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Generates I2Cx communication STOP condition.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C STOP condition generation. This parameter can be: ENABLE or DISABLE.

Return values

None.	
-------	--

I2C_Init()

```
void I2C_Init (
    I2C_TypeDef * I2Cx,
    I2C_InitTypeDef * I2C_InitStruct )
```

Initializes the I2Cx peripheral according to the specified parameters in the I2C_InitStruct.

Note

To use the I2C at 400 KHz (in fast mode), the PCLK1 frequency (I2C peripheral input clock) must be a multiple of 10 MHz.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_InitStruct</i>	pointer to a I2C_InitTypeDef structure that contains the configuration information for the specified I2C peripheral.

Return values

None	
------	--

I2C_NACKPositionConfig()

```
void I2C_NACKPositionConfig (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_NACKPosition )
```

Selects the specified I2C NACK position in master receiver mode.

Note

This function is useful in I2C Master Receiver mode when the number of data to be received is equal to 2. In this case, this function should be called (with parameter I2C_NACKPosition_Next) before data reception starts, as described in the 2-byte reception procedure recommended in Reference Manual in Section: Master receiver.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_NACKPosition</i>	specifies the NACK position. This parameter can be one of the following values: <ul style="list-style-type: none"> I2C_NACKPosition_Next: indicates that the next byte will be the last received byte. I2C_NACKPosition_Current: indicates that current byte is the last received byte.

Note

This function configures the same bit (POS) as [I2C_PECPositionConfig\(\)](#) but is intended to be used in I2C mode while [I2C_PECPositionConfig\(\)](#) is intended to be used in SMBUS mode.

Return values

<i>None</i>	
-------------	--

I2C_OwnAddress2Config()

```
void I2C_OwnAddress2Config (
    I2C_TypeDef * I2Cx,
    uint8_t Address )
```

Configures the specified I2C own address2.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>Address</i>	specifies the 7bit I2C own address2.

Return values

None.	
-------	--

I2C_Send7bitAddress()

```
void I2C_Send7bitAddress (
    I2C_TypeDef * I2Cx,
    uint8_t Address,
    uint8_t I2C_Direction )
```

Transmits the address byte to select the slave device.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>Address</i>	specifies the slave address which will be transmitted
<i>I2C_Direction</i>	specifies whether the I2C device will be a Transmitter or a Receiver. This parameter can be one of the following values <ul style="list-style-type: none"> • I2C_Direction_Transmitter: Transmitter mode • I2C_Direction_Receiver: Receiver mode

Return values

None.	
-------	--

I2C_SMBusAlertConfig()

```
void I2C_SMBusAlertConfig (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_SMBusAlert )
```

Drives the SMBusAlert pin high or low for the specified I2C.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_SMBusAlert</i>	specifies SMBAlert pin level. This parameter can be one of the following values: <ul style="list-style-type: none"> • I2C_SMBusAlert_Low: SMBAlert pin driven low • I2C_SMBusAlert_High: SMBAlert pin driven high

Return values

None	
------	--

I2C_SoftwareResetCmd()

```
void I2C_SoftwareResetCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C software reset.

Note

When software reset is enabled, the I2C IOs are released (this can be useful to recover from bus errors).

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C software reset. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

I2C_StretchClockCmd()

```
void I2C_StretchClockCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C Clock stretching.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2Cx Clock stretching. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

I2C_StructInit()

```
void I2C_StructInit (
    I2C_InitTypeDef * I2C_InitStruct )
```

Fills each I2C_InitStruct member with its default value.

Parameters

<i>I2C_InitStruct</i>	pointer to an I2C_InitTypeDef structure which will be initialized.
-----------------------	--

Return values

<i>None</i>	
-------------	--

4.1.4.4.3 Data transfers functions

Data transfers functions.

Functions

- void [I2C_SendData](#) (I2C_TypeDef *I2Cx, uint8_t Data)
Sends a data byte through the I2Cx peripheral.
- uint8_t [I2C_ReceiveData](#) (I2C_TypeDef *I2Cx)
Returns the most recent received data by the I2Cx peripheral.

4.1.4.4.3.1 Detailed Description

Data transfers functions.

```
=====
                        Data transfers functions
=====
```

4.1.4.4.3.2 Function Documentation

I2C_ReceiveData()

```
uint8_t I2C_ReceiveData (
    I2C_TypeDef * I2Cx )
```

Returns the most recent received data by the I2Cx peripheral.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
-------------	--

Return values

<i>The</i>	value of the received data.
------------	-----------------------------

I2C_SendData()

```
void I2C_SendData (
    I2C_TypeDef * I2Cx,
    uint8_t Data )
```

Sends a data byte through the I2Cx peripheral.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>Data</i>	Byte to be transmitted..

Return values

<i>None</i>	
-------------	--

4.1.4.4.4 PEC management functions

PEC management functions.

Functions

- void [I2C_TransmitPEC](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C PEC transfer.
- void [I2C_PECPositionConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_PECPosition)
Selects the specified I2C PEC position.
- void [I2C_CalculatePEC](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the PEC value calculation of the transferred bytes.
- uint8_t [I2C_GetPEC](#) (I2C_TypeDef *I2Cx)
Returns the PEC value for the specified I2C.

4.1.4.4.4.1 Detailed Description

PEC management functions.

```
=====
                        PEC management functions
=====
```

4.1.4.4.4.2 Function Documentation**I2C_CalculatePEC()**

```
void I2C_CalculatePEC (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the PEC value calculation of the transferred bytes.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2Cx PEC value calculation. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

I2C_GetPEC()

```
uint8_t I2C_GetPEC (
    I2C_TypeDef * I2Cx )
```

Returns the PEC value for the specified I2C.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
-------------	--

Return values

<i>The</i>	PEC value.
------------	------------

I2C_PECPositionConfig()

```
void I2C_PECPositionConfig (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_PECPosition )
```

Selects the specified I2C PEC position.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_PECPosition</i>	specifies the PEC position. This parameter can be one of the following values: <ul style="list-style-type: none"> I2C_PECPosition_Next: indicates that the next byte is PEC I2C_PECPosition_Current: indicates that current byte is PEC

Note

This function configures the same bit (POS) as [I2C_NACKPositionConfig\(\)](#) but is intended to be used in SMBUS mode while [I2C_NACKPositionConfig\(\)](#) is intended to be used in I2C mode.

Return values

<i>None</i>	
-------------	--

I2C_TransmitPEC()

```
void I2C_TransmitPEC (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C PEC transfer.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C PEC transmission. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.4.5 DMA transfers management functions

DMA transfers management functions.

Functions

- void [I2C_DMAMCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C DMA requests.
- void [I2C_DMALastTransferCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Specifies that the next DMA transfer is the last one.

4.1.4.4.5.1 Detailed Description

DMA transfers management functions.

```
=====
DMA transfers management functions
=====
This section provides functions allowing to configure the I2C DMA channels
requests.
```

4.1.4.4.5.2 Function Documentation**I2C_DMAMCmd()**

```
void I2C_DMAMCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Enables or disables the specified I2C DMA requests.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C DMA transfer. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

I2C_DMALastTransferCmd()

```
void I2C_DMALastTransferCmd (
    I2C_TypeDef * I2Cx,
    FunctionalState NewState )
```

Specifies that the next DMA transfer is the last one.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>NewState</i>	new state of the I2C DMA last transfer. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.4.4.6 Interrupts events and flags management functions

Interrupts, events and flags management functions.

Functions

- uint16_t [I2C_ReadRegister](#) (I2C_TypeDef *I2Cx, uint8_t I2C_Register)
Reads the specified I2C register and returns its value.
- void [I2C_ITConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_IT, FunctionalState NewState)
Enables or disables the specified I2C interrupts.
- ErrorStatus [I2C_CheckEvent](#) (I2C_TypeDef *I2Cx, uint32_t I2C_EVENT)
Checks whether the last I2Cx Event is equal to the one passed as parameter.
- uint32_t [I2C_GetLastEvent](#) (I2C_TypeDef *I2Cx)
Returns the last I2Cx Event.
- FlagStatus [I2C_GetFlagStatus](#) (I2C_TypeDef *I2Cx, uint32_t I2C_FLAG)
Checks whether the specified I2C flag is set or not.
- void [I2C_ClearFlag](#) (I2C_TypeDef *I2Cx, uint32_t I2C_FLAG)
Clears the I2Cx's pending flags.
- ITStatus [I2C_GetITStatus](#) (I2C_TypeDef *I2Cx, uint32_t I2C_IT)
Checks whether the specified I2C interrupt has occurred or not.
- void [I2C_ClearITPendingBit](#) (I2C_TypeDef *I2Cx, uint32_t I2C_IT)
Clears the I2Cx's interrupt pending bits.

4.1.4.4.6.1 Detailed Description

Interrupts, events and flags management functions.

```
=====
                        Interrupts, events and flags management functions
=====

This section provides functions allowing to configure the I2C Interrupts
sources and check or clear the flags or pending bits status.
The user should identify which mode will be used in his application to manage
the communication: Polling mode, Interrupt mode or DMA mode.

=====
                        I2C State Monitoring Functions
=====

This I2C driver provides three different ways for I2C state monitoring
depending on the application requirements and constraints:

1. Basic state monitoring (Using I2C_CheckEvent() function)
-----
It compares the status registers (SR1 and SR2) content to a given event
(can be the combination of one or more flags).
It returns SUCCESS if the current status includes the given flags
and returns ERROR if one or more flags are missing in the current status.

- When to use
  - This function is suitable for most applications as well as for startup
    activity since the events are fully described in the product reference
    manual (RM0090).
  - It is also suitable for users who need to define their own events.

- Limitations
  - If an error occurs (ie. error flags are set besides to the monitored
    flags), the I2C_CheckEvent() function may return SUCCESS despite
    the communication hold or corrupted real state.
    In this case, it is advised to use error interrupts to monitor
    the error events and handle them in the interrupt IRQ handler.

@note
For error management, it is advised to use the following functions:
- I2C_ITConfig() to configure and enable the error interrupts (I2C_IT_ERR).
- I2Cx_ER_IRQHandler() which is called when the error interrupt occurs.
  Where x is the peripheral instance (I2C1, I2C2 ...)
- I2C_GetFlagStatus() or I2C_GetITStatus() to be called into the
  I2Cx_ER_IRQHandler() function in order to determine which error occurred.
- I2C_ClearFlag() or I2C_ClearITPendingBit() and/or I2C_SoftwareResetCmd()
  and/or I2C_GenerateStop() in order to clear the error flag and source
  and return to correct communication status.

2. Advanced state monitoring (Using the function I2C_GetLastEvent())
-----
Using the function I2C_GetLastEvent() which returns the image of both status
registers in a single word (uint32_t) (Status Register 2 value is shifted left
by 16 bits and concatenated to Status Register 1).

- When to use
  - This function is suitable for the same applications above but it
    allows to overcome the mentioned limitation of I2C_GetFlagStatus()
    function.
  - The returned value could be compared to events already defined in
    the library (stm32f4xx_i2c.h) or to custom values defined by user.
    This function is suitable when multiple flags are monitored at the
    same time.
  - At the opposite of I2C_CheckEvent() function, this function allows
    user to choose when an event is accepted (when all events flags are
    set and no other flags are set or just when the needed flags are set
    like I2C_CheckEvent() function.

- Limitations
```


- User may need to define his own events.
- Same remark concerning the error management is applicable for this function if user decides to check only regular communication flags (and ignores error flags).

3. Flag-based state monitoring (Using the function I2C_GetFlagStatus())

Using the function I2C_GetFlagStatus() which simply returns the status of one single flag (ie. I2C_FLAG_RXNE ...).

- When to use
 - This function could be used for specific applications or in debug phase.
 - It is suitable when only one flag checking is needed (most I2C events are monitored through multiple flags).
- Limitations:
 - When calling this function, the Status register is accessed. Some flags are cleared when the status register is accessed. So checking the status of one Flag, may clear other ones.
 - Function may need to be called twice or more in order to monitor one single event.

For detailed description of Events, please refer to section I2C_Events in stm32f4xx_i2c.h file.

4.1.4.4.6.2 Function Documentation

I2C_CheckEvent()

```
ErrorStatus I2C_CheckEvent (
    I2C_TypeDef * I2Cx,
    uint32_t I2C_EVENT )
```

Checks whether the last I2Cx Event is equal to the one passed as parameter.

Parameters

I2Cx	where x can be 1, 2 or 3 to select the I2C peripheral.
------	--

Parameters

<i>I2C_EVENT</i>	<p>specifies the event to be checked. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED: EV1 • I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED: EV1 • I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED: EV1 • I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED: EV1 • I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED: EV1 • I2C_EVENT_SLAVE_BYTE_RECEIVED: EV2 • (I2C_EVENT_SLAVE_BYTE_RECEIVED I2C_FLAG_DUALF): EV2 • (I2C_EVENT_SLAVE_BYTE_RECEIVED I2C_FLAG_GENCALL): EV2 • I2C_EVENT_SLAVE_BYTE_TRANSMITTED: EV3 • (I2C_EVENT_SLAVE_BYTE_TRANSMITTED I2C_FLAG_DUALF): EV3 • (I2C_EVENT_SLAVE_BYTE_TRANSMITTED I2C_FLAG_GENCALL): EV3 • I2C_EVENT_SLAVE_ACK_FAILURE: EV3_2 • I2C_EVENT_SLAVE_STOP_DETECTED: EV4 • I2C_EVENT_MASTER_MODE_SELECT: EV5 • I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED: EV6 • I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED: EV6 • I2C_EVENT_MASTER_BYTE_RECEIVED: EV7 • I2C_EVENT_MASTER_BYTE_TRANSMITTING: EV8 • I2C_EVENT_MASTER_BYTE_TRANSMITTED: EV8_2 • I2C_EVENT_MASTER_MODE_ADDRESS10: EV9
------------------	---

Note

For detailed description of Events, please refer to section I2C_Events in [stm32f4xx_i2c.h](#) file.

Return values

<i>An</i>	<p>ErrorStatus enumeration value:</p> <ul style="list-style-type: none"> • SUCCESS: Last event is equal to the I2C_EVENT • ERROR: Last event is different from the I2C_EVENT
-----------	--

I2C_ClearFlag()

```
void I2C_ClearFlag (
```

```
I2C_TypeDef * I2Cx,
uint32_t I2C_FLAG )
```

Clears the I2Cx's pending flags.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_FLAG</i>	<p>specifies the flag to clear. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • I2C_FLAG_SMBALERT: SMBus Alert flag • I2C_FLAG_TIMEOUT: Timeout or Tlow error flag • I2C_FLAG_PECERR: PEC error in reception flag • I2C_FLAG_OVR: Overrun/Underrun flag (Slave mode) • I2C_FLAG_AF: Acknowledge failure flag • I2C_FLAG_ARLO: Arbitration lost flag (Master mode) • I2C_FLAG_BERR: Bus error flag

Note

STOPF (STOP detection) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetFlagStatus\(\)](#)) followed by a write operation to I2C_CR1 register ([I2C_Cmd\(\)](#)) to re-enable the I2C peripheral).

ADD10 (10-bit header sent) is cleared by software sequence: a read operation to I2C_SR1 ([I2C_GetFlagStatus\(\)](#)) followed by writing the second byte of the address in DR register.

BTF (Byte Transfer Finished) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetFlagStatus\(\)](#)) followed by a read/write to I2C_DR register ([I2C_SendData\(\)](#)).

ADDR (Address sent) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetFlagStatus\(\)](#)) followed by a read operation to I2C_SR2 register ((void)(I2Cx->SR2)).

SB (Start Bit) is cleared software sequence: a read operation to I2C_SR1 register ([I2C_GetFlagStatus\(\)](#)) followed by a write operation to I2C_DR register ([I2C_SendData\(\)](#)).

Return values

<i>None</i>	
-------------	--

I2C_ClearITPendingBit()

```
void I2C_ClearITPendingBit (
    I2C_TypeDef * I2Cx,
    uint32_t I2C_IT )
```

Clears the I2Cx's interrupt pending bits.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
-------------	--

Parameters

<i>I2C</i> ↔ <i>_IT</i>	<p>specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • I2C_IT_SMBALERT: SMBus Alert interrupt • I2C_IT_TIMEOUT: Timeout or Tlow error interrupt • I2C_IT_PECERR: PEC error in reception interrupt • I2C_IT_OVR: Overrun/Underrun interrupt (Slave mode) • I2C_IT_AF: Acknowledge failure interrupt • I2C_IT_ARLO: Arbitration lost interrupt (Master mode) • I2C_IT_BERR: Bus error interrupt
----------------------------	--

Note

STOPF (STOP detection) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetITStatus\(\)](#)) followed by a write operation to I2C_CR1 register ([I2C_Cmd\(\)](#)) to re-enable the I2C peripheral).

ADD10 (10-bit header sent) is cleared by software sequence: a read operation to I2C_SR1 ([I2C_GetITStatus\(\)](#)) followed by writing the second byte of the address in I2C_DR register.

BTF (Byte Transfer Finished) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetITStatus\(\)](#)) followed by a read/write to I2C_DR register ([I2C_SendData\(\)](#)).

ADDR (Address sent) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetITStatus\(\)](#)) followed by a read operation to I2C_SR2 register ((void)(I2Cx->SR2)).

SB (Start Bit) is cleared by software sequence: a read operation to I2C_SR1 register ([I2C_GetITStatus\(\)](#)) followed by a write operation to I2C_DR register ([I2C_SendData\(\)](#)).

Return values

<i>None</i>	
-------------	--

I2C_GetFlagStatus()

```
FlagStatus I2C_GetFlagStatus (
    I2C_TypeDef * I2Cx,
    uint32_t I2C_FLAG )
```

Checks whether the specified I2C flag is set or not.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
-------------	--

Parameters

<i>I2C_FLAG</i>	<p>specifies the flag to check. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • I2C_FLAG_DUALF: Dual flag (Slave mode) • I2C_FLAG_SMBHOST: SMBus host header (Slave mode) • I2C_FLAG_SMBDEFAULT: SMBus default header (Slave mode) • I2C_FLAG_GENCALL: General call header flag (Slave mode) • I2C_FLAG_TRA: Transmitter/Receiver flag • I2C_FLAG_BUSY: Bus busy flag • I2C_FLAG_MSL: Master/Slave flag • I2C_FLAG_SMBALERT: SMBus Alert flag • I2C_FLAG_TIMEOUT: Timeout or Tlow error flag • I2C_FLAG_PECERR: PEC error in reception flag • I2C_FLAG_OVR: Overrun/Underrun flag (Slave mode) • I2C_FLAG_AF: Acknowledge failure flag • I2C_FLAG_ARLO: Arbitration lost flag (Master mode) • I2C_FLAG_BERR: Bus error flag • I2C_FLAG_TXE: Data register empty flag (Transmitter) • I2C_FLAG_RXNE: Data register not empty (Receiver) flag • I2C_FLAG_STOPF: Stop detection flag (Slave mode) • I2C_FLAG_ADD10: 10-bit header sent flag (Master mode) • I2C_FLAG_BTF: Byte transfer finished flag • I2C_FLAG_ADDR: Address sent flag (Master mode) "ADSL" Address matched flag (Slave mode)"ENDAD" • I2C_FLAG_SB: Start bit flag (Master mode)
-----------------	--

Return values

<i>The</i>	new state of I2C_FLAG (SET or RESET).
------------	---------------------------------------

I2C_GetITStatus()

```
ITStatus I2C_GetITStatus (
    I2C_TypeDef * I2Cx,
    uint32_t I2C_IT )
```

Checks whether the specified I2C interrupt has occurred or not.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C↔ _IT</i>	<p>specifies the interrupt source to check. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • I2C_IT_SMBALERT: SMBus Alert flag • I2C_IT_TIMEOUT: Timeout or Tlow error flag • I2C_IT_PECERR: PEC error in reception flag • I2C_IT_OVR: Overrun/Underrun flag (Slave mode) • I2C_IT_AF: Acknowledge failure flag • I2C_IT_ARLO: Arbitration lost flag (Master mode) • I2C_IT_BERR: Bus error flag • I2C_IT_TXE: Data register empty flag (Transmitter) • I2C_IT_RXNE: Data register not empty (Receiver) flag • I2C_IT_STOPF: Stop detection flag (Slave mode) • I2C_IT_ADD10: 10-bit header sent flag (Master mode) • I2C_IT_BTF: Byte transfer finished flag • I2C_IT_ADDR: Address sent flag (Master mode) "ADSL" Address matched flag (Slave mode)"ENDAD" • I2C_IT_SB: Start bit flag (Master mode)

Return values

<i>The</i>	new state of I2C_IT (SET or RESET).
------------	-------------------------------------

I2C_GetLastEvent()

```
uint32_t I2C_GetLastEvent (
    I2C_TypeDef * I2Cx )
```

Returns the last I2Cx Event.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
-------------	--

Note

For detailed description of Events, please refer to section I2C_Events in [stm32f4xx_i2c.h](#) file.

Return values

<i>The</i>	last event
------------	------------

I2C_ITConfig()

```
void I2C_ITConfig (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_IT,
    FunctionalState NewState )
```

Enables or disables the specified I2C interrupts.

Parameters

<i>I2Cx</i>	where x can be 1, 2 or 3 to select the I2C peripheral.
<i>I2C_IT</i>	specifies the I2C interrupts sources to be enabled or disabled. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • I2C_IT_BUF: Buffer interrupt mask • I2C_IT_EVT: Event interrupt mask • I2C_IT_ERR: Error interrupt mask
<i>NewState</i>	new state of the specified I2C interrupts. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

I2C_ReadRegister()

```
uint16_t I2C_ReadRegister (
    I2C_TypeDef * I2Cx,
    uint8_t I2C_Register )
```

Reads the specified I2C register and returns its value.

Parameters

<i>I2C_Register</i>	specifies the register to read. This parameter can be one of the following values: <ul style="list-style-type: none"> • I2C_Register_CR1: CR1 register. • I2C_Register_CR2: CR2 register. • I2C_Register_OAR1: OAR1 register. • I2C_Register_OAR2: OAR2 register. • I2C_Register_DR: DR register. • I2C_Register_SR1: SR1 register. • I2C_Register_SR2: SR2 register. • I2C_Register_CCR: CCR register. • I2C_Register_TRISE: TRISE register.
---------------------	--

Return values

<i>The</i>	value of the read register.
------------	-----------------------------

4.1.4.5 I2C_Exported_Constants

Modules

- [I2C_mode](#)
- [I2C_duty_cycle_in_fast_mode](#)
- [I2C_acknowledgement](#)
- [I2C_transfer_direction](#)
- [I2C_acknowledged_address](#)
- [I2C_registers](#)
- [I2C_NACK_position](#)
- [I2C_SMBus_alert_pin_level](#)
- [I2C_PEC_position](#)
- [I2C_interrupts_definition](#)
- [I2C_flags_definition](#)
- [I2C_Events](#)
- [I2C_own_address1](#)
- [I2C_clock_speed](#)

Macros

- `#define IS_I2C_ALL_PERIPH(PERIPH)`

4.1.4.5.1 Detailed Description

4.1.4.5.2 Macro Definition Documentation

4.1.4.5.2.1 IS_I2C_ALL_PERIPH

```
#define IS_I2C_ALL_PERIPH(  
    PERIPH )
```

Value:

```
(( (PERIPH) == I2C1) || \  
 (PERIPH) == I2C2) || \  
 (PERIPH) == I2C3)
```

4.1.4.5.3 I2C_mode

Macros

- `#define I2C_Mode_I2C ((uint16_t)0x0000)`
- `#define I2C_Mode_SMBusDevice ((uint16_t)0x0002)`
- `#define I2C_Mode_SMBusHost ((uint16_t)0x000A)`
- `#define IS_I2C_MODE(MODE)`

4.1.4.5.3.1 Detailed Description

4.1.4.5.3.2 Macro Definition Documentation

I2C_Mode_I2C

```
#define I2C_Mode_I2C ((uint16_t)0x0000)
```

I2C_Mode_SMBusDevice

```
#define I2C_Mode_SMBusDevice ((uint16_t)0x0002)
```

I2C_Mode_SMBusHost

```
#define I2C_Mode_SMBusHost ((uint16_t)0x000A)
```

IS_I2C_MODE

```
#define IS_I2C_MODE(  
    MODE )
```

Value:

```
((MODE) == I2C_Mode_I2C) || \  
((MODE) == I2C_Mode_SMBusDevice) || \  
((MODE) == I2C_Mode_SMBusHost))
```

4.1.4.5.4 I2C_duty_cycle_in_fast_mode

Macros

- #define I2C_DutyCycle_16_9 ((uint16_t)0x4000)
- #define I2C_DutyCycle_2 ((uint16_t)0xBFFF)
- #define IS_I2C_DUTY_CYCLE(CYCLE)

4.1.4.5.4.1 Detailed Description

4.1.4.5.4.2 Macro Definition Documentation

I2C_DutyCycle_16_9

```
#define I2C_DutyCycle_16_9 ((uint16_t)0x4000)
```

I2C fast mode Tlow/Thigh = 16/9

I2C_DutyCycle_2

```
#define I2C_DutyCycle_2 ((uint16_t)0xBFFF)
```

I2C fast mode Tlow/Thigh = 2

IS_I2C_DUTY_CYCLE

```
#define IS_I2C_DUTY_CYCLE(  
    CYCLE )
```

Value:

```
((CYCLE) == I2C_DutyCycle_16_9) || \  
((CYCLE) == I2C_DutyCycle_2))
```

4.1.4.5.5 I2C_acknowledgement

Macros

- #define [I2C_Ack_Enable](#) ((uint16_t)0x0400)
- #define [I2C_Ack_Disable](#) ((uint16_t)0x0000)
- #define [IS_I2C_ACK_STATE](#)(STATE)

4.1.4.5.5.1 Detailed Description

4.1.4.5.5.2 Macro Definition Documentation

I2C_Ack_Disable

```
#define I2C_Ack_Disable ((uint16_t)0x0000)
```

I2C_Ack_Enable

```
#define I2C_Ack_Enable ((uint16_t)0x0400)
```

IS_I2C_ACK_STATE

```
#define IS_I2C_ACK_STATE(  
    STATE )
```

Value:

```
((STATE) == I2C_Ack_Enable) || \  
((STATE) == I2C_Ack_Disable))
```

4.1.4.5.6 I2C_transfer_direction

Macros

- #define I2C_Direction_Transmitter ((uint8_t)0x00)
- #define I2C_Direction_Receiver ((uint8_t)0x01)
- #define IS_I2C_DIRECTION(DIRECTION)

4.1.4.5.6.1 Detailed Description

4.1.4.5.6.2 Macro Definition Documentation

I2C_Direction_Receiver

```
#define I2C_Direction_Receiver ((uint8_t)0x01)
```

I2C_Direction_Transmitter

```
#define I2C_Direction_Transmitter ((uint8_t)0x00)
```

IS_I2C_DIRECTION

```
#define IS_I2C_DIRECTION(  
    DIRECTION )
```

Value:

```
(( (DIRECTION) == I2C_Direction_Transmitter) || \  
 (DIRECTION) == I2C_Direction_Receiver)
```

4.1.4.5.7 I2C_acknowledged_address

Macros

- #define I2C_AcknowledgedAddress_7bit ((uint16_t)0x4000)
- #define I2C_AcknowledgedAddress_10bit ((uint16_t)0xC000)
- #define IS_I2C_ACKNOWLEDGE_ADDRESS(ADDRESS)

4.1.4.5.7.1 Detailed Description

4.1.4.5.7.2 Macro Definition Documentation

I2C_AcknowledgedAddress_10bit

```
#define I2C_AcknowledgedAddress_10bit ((uint16_t)0xC000)
```

I2C_AcknowledgedAddress_7bit

```
#define I2C_AcknowledgedAddress_7bit ((uint16_t)0x4000)
```

IS_I2C_ACKNOWLEDGE_ADDRESS

```
#define IS_I2C_ACKNOWLEDGE_ADDRESS(  
    ADDRESS )
```

Value:

```
((ADDRESS) == I2C_AcknowledgedAddress_7bit) || \  
((ADDRESS) == I2C_AcknowledgedAddress_10bit))
```

4.1.4.5.8 I2C_registers**Macros**

- `#define I2C_Register_CR1 ((uint8_t)0x00)`
- `#define I2C_Register_CR2 ((uint8_t)0x04)`
- `#define I2C_Register_OAR1 ((uint8_t)0x08)`
- `#define I2C_Register_OAR2 ((uint8_t)0x0C)`
- `#define I2C_Register_DR ((uint8_t)0x10)`
- `#define I2C_Register_SR1 ((uint8_t)0x14)`
- `#define I2C_Register_SR2 ((uint8_t)0x18)`
- `#define I2C_Register_CCR ((uint8_t)0x1C)`
- `#define I2C_Register_TRISE ((uint8_t)0x20)`
- `#define IS_I2C_REGISTER(REGISTER)`

4.1.4.5.8.1 Detailed Description**4.1.4.5.8.2 Macro Definition Documentation****I2C_Register_CCR**

```
#define I2C_Register_CCR ((uint8_t)0x1C)
```

I2C_Register_CR1

```
#define I2C_Register_CR1 ((uint8_t)0x00)
```

I2C_Register_CR2

```
#define I2C_Register_CR2 ((uint8_t)0x04)
```

I2C_Register_DR

```
#define I2C_Register_DR ((uint8_t)0x10)
```

I2C_Register_OAR1

```
#define I2C_Register_OAR1 ((uint8_t)0x08)
```

I2C_Register_OAR2

```
#define I2C_Register_OAR2 ((uint8_t)0x0C)
```

I2C_Register_SR1

```
#define I2C_Register_SR1 ((uint8_t)0x14)
```

I2C_Register_SR2

```
#define I2C_Register_SR2 ((uint8_t)0x18)
```

I2C_Register_TRISE

```
#define I2C_Register_TRISE ((uint8_t)0x20)
```

IS_I2C_REGISTER

```
#define IS_I2C_REGISTER(  
    REGISTER )
```

Value:

```
((REGISTER) == I2C_Register_CR1) || \  
(REGISTER) == I2C_Register_CR2) || \  
(REGISTER) == I2C_Register_OAR1) || \  
(REGISTER) == I2C_Register_OAR2) || \  
(REGISTER) == I2C_Register_DR) || \  
(REGISTER) == I2C_Register_SR1) || \  
(REGISTER) == I2C_Register_SR2) || \  
(REGISTER) == I2C_Register_CCR) || \  
(REGISTER) == I2C_Register_TRISE))
```

4.1.4.5.9 I2C_NACK_position**Macros**

- `#define I2C_NACKPosition_Next ((uint16_t)0x0800)`
- `#define I2C_NACKPosition_Current ((uint16_t)0xF7FF)`
- `#define IS_I2C_NACK_POSITION(POSITION)`

4.1.4.5.9.1 Detailed Description**4.1.4.5.9.2 Macro Definition Documentation****I2C_NACKPosition_Current**

```
#define I2C_NACKPosition_Current ((uint16_t)0xF7FF)
```

I2C_NACKPosition_Next

```
#define I2C_NACKPosition_Next ((uint16_t)0x0800)
```

IS_I2C_NACK_POSITION

```
#define IS_I2C_NACK_POSITION(  
    POSITION )
```

Value:

```
((POSITION) == I2C_NACKPosition_Next) || \  
((POSITION) == I2C_NACKPosition_Current))
```

4.1.4.5.10 I2C_SMBus_alert_pin_level**Macros**

- #define [I2C_SMBusAlert_Low](#) ((uint16_t)0x2000)
- #define [I2C_SMBusAlert_High](#) ((uint16_t)0xDFFF)
- #define [IS_I2C_SMBUS_ALERT](#)(ALERT)

4.1.4.5.10.1 Detailed Description**4.1.4.5.10.2 Macro Definition Documentation****I2C_SMBusAlert_High**

```
#define I2C_SMBusAlert_High ((uint16_t)0xDFFF)
```

I2C_SMBusAlert_Low

```
#define I2C_SMBusAlert_Low ((uint16_t)0x2000)
```

IS_I2C_SMBUS_ALERT

```
#define IS_I2C_SMBUS_ALERT(  
    ALERT )
```

Value:

```
((ALERT) == I2C_SMBusAlert_Low) || \  
((ALERT) == I2C_SMBusAlert_High))
```

4.1.4.5.11 I2C_PEC_position**Macros**

- #define [I2C_PECPosition_Next](#) ((uint16_t)0x0800)
- #define [I2C_PECPosition_Current](#) ((uint16_t)0xF7FF)
- #define [IS_I2C_PEC_POSITION](#)(POSITION)

4.1.4.5.11.1 Detailed Description

4.1.4.5.11.2 Macro Definition Documentation

I2C_PECPosition_Current

```
#define I2C_PECPosition_Current ((uint16_t)0xF7FF)
```

I2C_PECPosition_Next

```
#define I2C_PECPosition_Next ((uint16_t)0x0800)
```

IS_I2C_PEC_POSITION

```
#define IS_I2C_PEC_POSITION(  
    POSITION )
```

Value:

```
((POSITION) == I2C_PECPosition_Next) || \  
(POSITION) == I2C_PECPosition_Current)
```

4.1.4.5.12 I2C_interrupts_definition

Macros

- #define I2C_IT_BUF ((uint16_t)0x0400)
- #define I2C_IT_EVT ((uint16_t)0x0200)
- #define I2C_IT_ERR ((uint16_t)0x0100)
- #define IS_I2C_CONFIG_IT(IT) (((IT) & (uint16_t)0xF8FF) == 0x00) && ((IT) != 0x00)
- #define I2C_IT_SMBALERT ((uint32_t)0x01008000)
- #define I2C_IT_TIMEOUT ((uint32_t)0x01004000)
- #define I2C_IT_PECERR ((uint32_t)0x01001000)
- #define I2C_IT_OVR ((uint32_t)0x01000800)
- #define I2C_IT_AF ((uint32_t)0x01000400)
- #define I2C_IT_ARLO ((uint32_t)0x01000200)
- #define I2C_IT_BERR ((uint32_t)0x01000100)
- #define I2C_IT_TXE ((uint32_t)0x06000080)
- #define I2C_IT_RXNE ((uint32_t)0x06000040)
- #define I2C_IT_STOPF ((uint32_t)0x02000010)
- #define I2C_IT_ADD10 ((uint32_t)0x02000008)
- #define I2C_IT_BTF ((uint32_t)0x02000004)
- #define I2C_IT_ADDR ((uint32_t)0x02000002)
- #define I2C_IT_SB ((uint32_t)0x02000001)
- #define IS_I2C_CLEAR_IT(IT) (((IT) & (uint16_t)0x20FF) == 0x00) && ((IT) != (uint16_t)0x00)
- #define IS_I2C_GET_IT(IT)

4.1.4.5.12.1 Detailed Description

4.1.4.5.12.2 Macro Definition Documentation

I2C_IT_ADD10

```
#define I2C_IT_ADD10 ((uint32_t)0x02000008)
```

I2C_IT_ADDR

```
#define I2C_IT_ADDR ((uint32_t)0x02000002)
```

I2C_IT_AF

```
#define I2C_IT_AF ((uint32_t)0x01000400)
```

I2C_IT_ARLO

```
#define I2C_IT_ARLO ((uint32_t)0x01000200)
```

I2C_IT_BERR

```
#define I2C_IT_BERR ((uint32_t)0x01000100)
```

I2C_IT_BTF

```
#define I2C_IT_BTF ((uint32_t)0x02000004)
```

I2C_IT_BUF

```
#define I2C_IT_BUF ((uint16_t)0x0400)
```

I2C_IT_ERR

```
#define I2C_IT_ERR ((uint16_t)0x0100)
```

I2C_IT_EVT

```
#define I2C_IT_EVT ((uint16_t)0x0200)
```


I2C_IT_OVR

```
#define I2C_IT_OVR ((uint32_t)0x01000800)
```

I2C_IT_PECERR

```
#define I2C_IT_PECERR ((uint32_t)0x01001000)
```

I2C_IT_RXNE

```
#define I2C_IT_RXNE ((uint32_t)0x06000040)
```

I2C_IT_SB

```
#define I2C_IT_SB ((uint32_t)0x02000001)
```

I2C_IT_SMBALERT

```
#define I2C_IT_SMBALERT ((uint32_t)0x01008000)
```

I2C_IT_STOPF

```
#define I2C_IT_STOPF ((uint32_t)0x02000010)
```

I2C_IT_TIMEOUT

```
#define I2C_IT_TIMEOUT ((uint32_t)0x01004000)
```

I2C_IT_TXE

```
#define I2C_IT_TXE ((uint32_t)0x06000080)
```

IS_I2C_CLEAR_IT

```
#define IS_I2C_CLEAR_IT(  
    IT ) (((IT) & (uint16_t)0x20FF) == 0x00) && ((IT) != (uint16_t)0x00)
```

IS_I2C_CONFIG_IT

```
#define IS_I2C_CONFIG_IT(  
    IT ) (((IT) & (uint16_t)0xF8FF) == 0x00) && ((IT) != 0x00)
```

IS_I2C_GET_IT

```
#define IS_I2C_GET_IT(  
    IT )
```

Value:

```
((IT) == I2C_IT_SMBALERT) || ((IT) == I2C_IT_TIMEOUT) || \  
((IT) == I2C_IT_PECERR) || ((IT) == I2C_IT_OVR) || \  
((IT) == I2C_IT_AF) || ((IT) == I2C_IT_ARLO) || \  
((IT) == I2C_IT_BERR) || ((IT) == I2C_IT_TXE) || \  
((IT) == I2C_IT_RXNE) || ((IT) == I2C_IT_STOPF) || \  
((IT) == I2C_IT_ADD10) || ((IT) == I2C_IT_BTIF) || \  
((IT) == I2C_IT_ADDR) || ((IT) == I2C_IT_SB))
```

4.1.4.5.13 I2C_flags_definition

Macros

- #define **I2C_FLAG_DUALF** ((uint32_t)0x00800000)
SR2 register flags
- #define **I2C_FLAG_SMBHOST** ((uint32_t)0x00400000)
- #define **I2C_FLAG_SMBDEFAULT** ((uint32_t)0x00200000)
- #define **I2C_FLAG_GENCALL** ((uint32_t)0x00100000)
- #define **I2C_FLAG_TRA** ((uint32_t)0x00040000)
- #define **I2C_FLAG_BUSY** ((uint32_t)0x00020000)
- #define **I2C_FLAG_MSL** ((uint32_t)0x00010000)
- #define **I2C_FLAG_SMBALERT** ((uint32_t)0x10008000)
SR1 register flags
- #define **I2C_FLAG_TIMEOUT** ((uint32_t)0x10004000)
- #define **I2C_FLAG_PECERR** ((uint32_t)0x10001000)
- #define **I2C_FLAG_OVR** ((uint32_t)0x10000800)
- #define **I2C_FLAG_AF** ((uint32_t)0x10000400)
- #define **I2C_FLAG_ARLO** ((uint32_t)0x10000200)
- #define **I2C_FLAG_BERR** ((uint32_t)0x10000100)
- #define **I2C_FLAG_TXE** ((uint32_t)0x10000080)
- #define **I2C_FLAG_RXNE** ((uint32_t)0x10000040)
- #define **I2C_FLAG_STOPF** ((uint32_t)0x10000010)
- #define **I2C_FLAG_ADD10** ((uint32_t)0x10000008)
- #define **I2C_FLAG_BTIF** ((uint32_t)0x10000004)
- #define **I2C_FLAG_ADDR** ((uint32_t)0x10000002)
- #define **I2C_FLAG_SB** ((uint32_t)0x10000001)
- #define **IS_I2C_CLEAR_FLAG**(FLAG) (((FLAG) & (uint16_t)0x20FF) == 0x00) && ((FLAG) != (uint16_t)0x00)
- #define **IS_I2C_GET_FLAG**(FLAG)

4.1.4.5.13.1 Detailed Description

4.1.4.5.13.2 Macro Definition Documentation

I2C_FLAG_ADD10

```
#define I2C_FLAG_ADD10 ((uint32_t)0x10000008)
```

I2C_FLAG_ADDR

```
#define I2C_FLAG_ADDR ((uint32_t)0x10000002)
```

I2C_FLAG_AF

```
#define I2C_FLAG_AF ((uint32_t)0x10000400)
```

I2C_FLAG_ARLO

```
#define I2C_FLAG_ARLO ((uint32_t)0x10000200)
```

I2C_FLAG_BERR

```
#define I2C_FLAG_BERR ((uint32_t)0x10000100)
```

I2C_FLAG_BTF

```
#define I2C_FLAG_BTF ((uint32_t)0x10000004)
```

I2C_FLAG_BUSY

```
#define I2C_FLAG_BUSY ((uint32_t)0x00020000)
```

I2C_FLAG_DUALF

```
#define I2C_FLAG_DUALF ((uint32_t)0x00800000)
```

SR2 register flags

I2C_FLAG_GENCALL

```
#define I2C_FLAG_GENCALL ((uint32_t)0x00100000)
```

I2C_FLAG_MSL

```
#define I2C_FLAG_MSL ((uint32_t)0x00010000)
```

I2C_FLAG_OVR

```
#define I2C_FLAG_OVR ((uint32_t)0x10000800)
```

I2C_FLAG_PECERR

```
#define I2C_FLAG_PECERR ((uint32_t)0x10001000)
```

I2C_FLAG_RXNE

```
#define I2C_FLAG_RXNE ((uint32_t)0x10000040)
```

I2C_FLAG_SB

```
#define I2C_FLAG_SB ((uint32_t)0x10000001)
```

I2C_FLAG_SMBALERT

```
#define I2C_FLAG_SMBALERT ((uint32_t)0x10008000)
```

SR1 register flags

I2C_FLAG_SMBDEFAULT

```
#define I2C_FLAG_SMBDEFAULT ((uint32_t)0x00200000)
```

I2C_FLAG_SMBHOST

```
#define I2C_FLAG_SMBHOST ((uint32_t)0x00400000)
```

I2C_FLAG_STOPF

```
#define I2C_FLAG_STOPF ((uint32_t)0x10000010)
```

I2C_FLAG_TIMEOUT

```
#define I2C_FLAG_TIMEOUT ((uint32_t)0x10004000)
```

I2C_FLAG_TRA

```
#define I2C_FLAG_TRA ((uint32_t)0x00040000)
```

I2C_FLAG_TXE

```
#define I2C_FLAG_TXE ((uint32_t)0x10000080)
```

IS_I2C_CLEAR_FLAG

```
#define IS_I2C_CLEAR_FLAG(  
    FLAG ) (((FLAG) & (uint16_t)0x20FF) == 0x00) && ((FLAG) != (uint16_t)0x00)
```

IS_I2C_GET_FLAG

```
#define IS_I2C_GET_FLAG(  
    FLAG )
```

Value:

```
((FLAG) == I2C_FLAG_DUALF) || ((FLAG) == I2C_FLAG_SMBHOST) || \  
((FLAG) == I2C_FLAG_SMBDEFAULT) || ((FLAG) == I2C_FLAG_GENCALL) || \  
((FLAG) == I2C_FLAG_TRA) || ((FLAG) == I2C_FLAG_BUSY) || \  
((FLAG) == I2C_FLAG_MSL) || ((FLAG) == I2C_FLAG_SMBALERT) || \  
((FLAG) == I2C_FLAG_TIMEOUT) || ((FLAG) == I2C_FLAG_PECERR) || \  
((FLAG) == I2C_FLAG_OVR) || ((FLAG) == I2C_FLAG_AF) || \  
((FLAG) == I2C_FLAG_ARLO) || ((FLAG) == I2C_FLAG_BERR) || \  
((FLAG) == I2C_FLAG_TXE) || ((FLAG) == I2C_FLAG_RXNE) || \  
((FLAG) == I2C_FLAG_STOPF) || ((FLAG) == I2C_FLAG_ADD10) || \  
((FLAG) == I2C_FLAG_BTF) || ((FLAG) == I2C_FLAG_ADDR) || \  
((FLAG) == I2C_FLAG_SB)
```

4.1.4.5.14 I2C_Events**Macros**

- #define **I2C_EVENT_MASTER_MODE_SELECT** ((uint32_t)0x00030001) /* BUSY, MSL and SB flag */
Communication start.
- #define **I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED** ((uint32_t)0x00070082) /* BUSY, MSL, ADDR, TXE and TRA flags */
Address Acknowledge.
- #define **I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED** ((uint32_t)0x00030002) /* BUSY, MSL and ADDR flags */
- #define **I2C_EVENT_MASTER_MODE_ADDRESS10** ((uint32_t)0x00030008) /* BUSY, MSL and ADD10 flags */
- #define **I2C_EVENT_MASTER_BYTE_RECEIVED** ((uint32_t)0x00030040) /* BUSY, MSL and RXNE flags */
Communication events.
- #define **I2C_EVENT_MASTER_BYTE_TRANSMITTING** ((uint32_t)0x00070080) /* TRA, BUSY, MSL, TXE flags */
- #define **I2C_EVENT_MASTER_BYTE_TRANSMITTED** ((uint32_t)0x00070084) /* TRA, BUSY, MSL, TXE and BTF flags */
- #define **I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED** ((uint32_t)0x00020002) /* BUSY and ADDR flags */
Communication start events.
- #define **I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED** ((uint32_t)0x00060082) /* TRA, BUSY, TXE and ADDR flags */
- #define **I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED** ((uint32_t)0x00820000) /* DUALF and BUSY flags */
- #define **I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED** ((uint32_t)0x00860080) /* DUALF, TRA, BUSY and TXE flags */
- #define **I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED** ((uint32_t)0x00120000) /* GENCALL and BUSY flags */
- #define **I2C_EVENT_SLAVE_BYTE_RECEIVED** ((uint32_t)0x00020040) /* BUSY and RXNE flags */
Communication events.

- `#define I2C_EVENT_SLAVE_STOP_DETECTED ((uint32_t)0x00000010) /* STOPF flag */`
- `#define I2C_EVENT_SLAVE_BYTE_TRANSMITTED ((uint32_t)0x00060084) /* TRA, BUSY, TXE and BTF flags */`
- `#define I2C_EVENT_SLAVE_BYTE_TRANSMITTING ((uint32_t)0x00060080) /* TRA, BUSY and TXE flags */`
- `#define I2C_EVENT_SLAVE_ACK_FAILURE ((uint32_t)0x00000400) /* AF flag */`
- `#define IS_I2C_EVENT(EVENT)`

4.1.4.5.14.1 Detailed Description

4.1.4.5.14.2 Macro Definition Documentation

I2C_EVENT_MASTER_BYTE_RECEIVED

```
#define I2C_EVENT_MASTER_BYTE_RECEIVED ((uint32_t)0x00030040) /* BUSY, MSL and RXNE flags */
```

Communication events.

If a communication is established (START condition generated and slave address acknowledged) then the master has to check on one of the following events for communication procedures:

1) Master Receiver mode: The master has to wait on the event EV7 then to read the data received from the slave ([I2C_ReceiveData\(\)](#) function).

2) Master Transmitter mode: The master has to send data ([I2C_SendData\(\)](#) function) then to wait on event EV8 or EV8_2. These two events are similar:

- EV8 means that the data has been written in the data register and is being shifted out.
- EV8_2 means that the data has been physically shifted out and output on the bus. In most cases, using EV8 is sufficient for the application. Using EV8_2 leads to a slower communication but ensure more reliable test. EV8_2 is also more suitable than EV8 for testing on the last data transmission (before Stop condition generation).

Note

In case the user software does not guarantee that this event EV7 is managed before the current byte end of transfer, then user may check on EV7 and BTF flag at the same time (ie. `(I2C_EVENT_MASTER_BYTE_RECEIVED | I2C_FLAG_BTF)`). In this case the communication may be slower.

I2C_EVENT_MASTER_BYTE_TRANSMITTED

```
#define I2C_EVENT_MASTER_BYTE_TRANSMITTED ((uint32_t)0x00070084) /* TRA, BUSY, MSL, TXE and BTF flags */
```

I2C_EVENT_MASTER_BYTE_TRANSMITTING

```
#define I2C_EVENT_MASTER_BYTE_TRANSMITTING ((uint32_t)0x00070080) /* TRA, BUSY, MSL, TXE flags */
```

I2C_EVENT_MASTER_MODE_ADDRESS10

```
#define I2C_EVENT_MASTER_MODE_ADDRESS10 ((uint32_t)0x00030008) /* BUSY, MSL and ADD10 flags */
```

I2C_EVENT_MASTER_MODE_SELECT

```
#define I2C_EVENT_MASTER_MODE_SELECT ((uint32_t)0x00030001) /* BUSY, MSL and SB flag */
```

Communication start.

4.1.4.5.14.3 I2C Master Events (Events grouped in order of communication)

After sending the START condition ([I2C_GenerateSTART\(\)](#) function) the master has to wait for this event. It means that the Start condition has been correctly released on the I2C bus (the bus is free, no other devices is communicating).

I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED

```
#define I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED ((uint32_t)0x00030002) /* BUSY, MSL and ADDR flags */
```

I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED

```
#define I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED ((uint32_t)0x00070082) /* BUSY, MSL, ADDR, TXE and TRA flags */
```

Address Acknowledge.

After checking on EV5 (start condition correctly released on the bus), the master sends the address of the slave(s) with which it will communicate ([I2C_Send7bitAddress\(\)](#) function, it also determines the direction of the communication: Master transmitter or Receiver). Then the master has to wait that a slave acknowledges his address. If an acknowledge is sent on the bus, one of the following events will be set:

- 1) In case of Master Receiver (7-bit addressing): the I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED event is set.
- 2) In case of Master Transmitter (7-bit addressing): the I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED is set
- 3) In case of 10-Bit addressing mode, the master (just after generating the START and checking on EV5) has to send the header of 10-bit addressing mode ([I2C_SendData\(\)](#) function). Then master should wait on EV9. It means that the 10-bit addressing header has been correctly sent on the bus. Then master should send the second part of the 10-bit address (LSB) using the function [I2C_Send7bitAddress\(\)](#). Then master should wait for event EV6.

I2C_EVENT_SLAVE_ACK_FAILURE

```
#define I2C_EVENT_SLAVE_ACK_FAILURE ((uint32_t)0x00000400) /* AF flag */
```

I2C_EVENT_SLAVE_BYTE_RECEIVED

```
#define I2C_EVENT_SLAVE_BYTE_RECEIVED ((uint32_t)0x00020040) /* BUSY and RXNE flags */
```

Communication events.

Wait on one of these events when EV1 has already been checked and:

- Slave RECEIVER mode:
 - EV2: When the application is expecting a data byte to be received.
 - EV4: When the application is expecting the end of the communication: master sends a stop condition and data transmission is stopped.
- Slave Transmitter mode:

- EV3: When a byte has been transmitted by the slave and the application is expecting the end of the byte transmission. The two events `I2C_EVENT_SLAVE_BYTE_TRANSMITTED` and `I2C_EVENT_SLAVE_BYTE_TRANSMITTING` are similar. The second one can optionally be used when the user software doesn't guarantee the EV3 is managed before the current byte end of transfer.
- EV3_2: When the master sends a NACK in order to tell slave that data transmission shall end (before sending the STOP condition). In this case slave has to stop sending data bytes and expect a Stop condition on the bus.

Note

In case the user software does not guarantee that the event EV2 is managed before the current byte end of transfer, then user may check on EV2 and BTF flag at the same time (ie. `(I2C_EVENT_SLAVE_BYTE_RECEIVED | I2C_FLAG_BTF)`). In this case the communication may be slower.

I2C_EVENT_SLAVE_BYTE_TRANSMITTED

```
#define I2C_EVENT_SLAVE_BYTE_TRANSMITTED ((uint32_t)0x00060084) /* TRA, BUSY, TXE and BTF flags */
```

I2C_EVENT_SLAVE_BYTE_TRANSMITTING

```
#define I2C_EVENT_SLAVE_BYTE_TRANSMITTING ((uint32_t)0x00060080) /* TRA, BUSY and TXE flags */
```

I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED

```
#define I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED ((uint32_t)0x00120000) /* GENCALL and BUSY flags */
```

I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED

```
#define I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED ((uint32_t)0x00020002) /* BUSY and ADDR flags */
```

Communication start events.

4.1.4.5.14.4 I2C Slave Events (Events grouped in order of communication)

Wait on one of these events at the start of the communication. It means that the I2C peripheral detected a Start condition on the bus (generated by master device) followed by the peripheral address. The peripheral generates an ACK condition on the bus (if the acknowledge feature is enabled through function [I2C_AcknowledgeConfig\(\)](#)) and the events listed above are set :

- 1) In normal case (only one address managed by the slave), when the address sent by the master matches the own address of the peripheral (configured by `I2C_OwnAddress1` field) the `I2C_EVENT_SLAVE_XXX_ADDRESS_MATCHED` event is set (where XXX could be TRANSMITTER or RECEIVER).
- 2) In case the address sent by the master matches the second address of the peripheral (configured by the function [I2C_OwnAddress2Config\(\)](#) and enabled by the function [I2C_DualAddressCmd\(\)](#)) the events `I2C_EVENT_SLAVE_XXX_SECONDADDRESS_MATCHED` (where XXX could be TRANSMITTER or RECEIVER) are set.
- 3) In case the address sent by the master is General Call (address 0x00) and if the General Call is enabled for the peripheral (using function [I2C_GeneralCallCmd\(\)](#)) the following event is set `I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED`.

I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED

```
#define I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED ((uint32_t)0x00820000) /* DUALF and BUSY flags */
```

I2C_EVENT_SLAVE_STOP_DETECTED

```
#define I2C_EVENT_SLAVE_STOP_DETECTED ((uint32_t)0x00000010) /* STOPF flag */
```


I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED

```
#define I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED ((uint32_t)0x00060082) /* TRA, BUSY, TXE
and ADDR flags */
```

I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED

```
#define I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED ((uint32_t)0x00860080) /* DUALF,
TRA, BUSY and TXE flags */
```

IS_I2C_EVENT

```
#define IS_I2C_EVENT(
    EVENT )
```

Value:

```
((EVENT) == I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED) || \
(EVENT) == I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED) || \
(EVENT) == I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED) || \
(EVENT) == I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED) || \
(EVENT) == I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED) || \
(EVENT) == I2C_EVENT_SLAVE_BYTE_RECEIVED) || \
(EVENT) == (I2C_EVENT_SLAVE_BYTE_RECEIVED | I2C_FLAG_DUALF)) || \
(EVENT) == (I2C_EVENT_SLAVE_BYTE_RECEIVED | I2C_FLAG_GENCALL)) || \
(EVENT) == I2C_EVENT_SLAVE_BYTE_TRANSMITTED) || \
(EVENT) == (I2C_EVENT_SLAVE_BYTE_TRANSMITTED | I2C_FLAG_DUALF)) || \
(EVENT) == (I2C_EVENT_SLAVE_BYTE_TRANSMITTED | I2C_FLAG_GENCALL)) || \
(EVENT) == I2C_EVENT_SLAVE_STOP_DETECTED) || \
(EVENT) == I2C_EVENT_MASTER_MODE_SELECT) || \
(EVENT) == I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED) || \
(EVENT) == I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED) || \
(EVENT) == I2C_EVENT_MASTER_BYTE_RECEIVED) || \
(EVENT) == I2C_EVENT_MASTER_BYTE_TRANSMITTED) || \
(EVENT) == I2C_EVENT_MASTER_BYTE_TRANSMITTING) || \
(EVENT) == I2C_EVENT_MASTER_MODE_ADDRESS10) || \
(EVENT) == I2C_EVENT_SLAVE_ACK_FAILURE))
```

4.1.4.5.15 I2C_own_address1**Macros**

- #define [IS_I2C_OWN_ADDRESS1](#)(ADDRESS1) ((ADDRESS1) <= 0x3FF)

4.1.4.5.15.1 Detailed Description**4.1.4.5.15.2 Macro Definition Documentation****IS_I2C_OWN_ADDRESS1**

```
#define IS_I2C_OWN_ADDRESS1(
    ADDRESS1 ) ((ADDRESS1) <= 0x3FF)
```

4.1.4.5.16 I2C_clock_speed**Macros**

- #define [IS_I2C_CLOCK_SPEED](#)(SPEED) (((SPEED) >= 0x1) && ((SPEED) <= 400000))

4.1.4.5.16.1 Detailed Description**4.1.4.5.16.2 Macro Definition Documentation****IS_I2C_CLOCK_SPEED**

```
#define IS_I2C_CLOCK_SPEED(
    SPEED ) (((SPEED) >= 0x1) && ((SPEED) <= 400000))
```

4.1.5 RCC

RCC driver modules.

Modules

- [RCC_Private_Functions](#)
- [RCC_Exported_Constants](#)

Data Structures

- struct [RCC_ClocksTypeDef](#)

Macros

- `#define` [RCC_OFFSET](#) (RCC_BASE - PERIPH_BASE)
- `#define` [CR_OFFSET](#) ([RCC_OFFSET](#) + 0x00)
- `#define` [HSION_BitNumber](#) 0x00
- `#define` [CR_HSION_BB](#) (PERIPH_BB_BASE + ([CR_OFFSET](#) * 32) + ([HSION_BitNumber](#) * 4))
- `#define` [CSSON_BitNumber](#) 0x13
- `#define` [CR_CSSON_BB](#) (PERIPH_BB_BASE + ([CR_OFFSET](#) * 32) + ([CSSON_BitNumber](#) * 4))
- `#define` [PLLON_BitNumber](#) 0x18
- `#define` [CR_PLLON_BB](#) (PERIPH_BB_BASE + ([CR_OFFSET](#) * 32) + ([PLLON_BitNumber](#) * 4))
- `#define` [PLLI2SON_BitNumber](#) 0x1A
- `#define` [CR_PLLI2SON_BB](#) (PERIPH_BB_BASE + ([CR_OFFSET](#) * 32) + ([PLLI2SON_BitNumber](#) * 4))
- `#define` [CFGR_OFFSET](#) ([RCC_OFFSET](#) + 0x08)
- `#define` [I2SSRC_BitNumber](#) 0x17
- `#define` [CFGR_I2SSRC_BB](#) (PERIPH_BB_BASE + ([CFGR_OFFSET](#) * 32) + ([I2SSRC_BitNumber](#) * 4))
- `#define` [BDCR_OFFSET](#) ([RCC_OFFSET](#) + 0x70)
- `#define` [RTCEN_BitNumber](#) 0x0F
- `#define` [BDCR_RTCEN_BB](#) (PERIPH_BB_BASE + ([BDCR_OFFSET](#) * 32) + ([RTCEN_BitNumber](#) * 4))
- `#define` [BDRST_BitNumber](#) 0x10
- `#define` [BDCR_BDRST_BB](#) (PERIPH_BB_BASE + ([BDCR_OFFSET](#) * 32) + ([BDRST_BitNumber](#) * 4))
- `#define` [CSR_OFFSET](#) ([RCC_OFFSET](#) + 0x74)
- `#define` [LSION_BitNumber](#) 0x00
- `#define` [CSR_LSION_BB](#) (PERIPH_BB_BASE + ([CSR_OFFSET](#) * 32) + ([LSION_BitNumber](#) * 4))
- `#define` [CFGR_MCO2_RESET_MASK](#) ((uint32_t)0x07FFFFFF)
- `#define` [CFGR_MCO1_RESET_MASK](#) ((uint32_t)0xF89FFFFFF)
- `#define` [FLAG_MASK](#) ((uint8_t)0x1F)
- `#define` [CR_BYTE3_ADDRESS](#) ((uint32_t)0x40023802)
- `#define` [CIR_BYTE2_ADDRESS](#) ((uint32_t)(RCC_BASE + 0x0C + 0x01))
- `#define` [CIR_BYTE3_ADDRESS](#) ((uint32_t)(RCC_BASE + 0x0C + 0x02))
- `#define` [BDCR_ADDRESS](#) (PERIPH_BASE + [BDCR_OFFSET](#))

Functions

- void [RCC_DeInit](#) (void)
Resets the RCC clock configuration to the default reset state.
- void [RCC_HSEConfig](#) (uint8_t RCC_HSE)
Configures the External High Speed oscillator (HSE).
- ErrorStatus [RCC_WaitForHSEStartUp](#) (void)
Waits for HSE start-up.
- void [RCC_AdjustHSICalibrationValue](#) (uint8_t HSICalibrationValue)
Adjusts the Internal High Speed oscillator (HSI) calibration value.
- void [RCC_HSICmd](#) (FunctionalState NewState)
Enables or disables the Internal High Speed oscillator (HSI).
- void [RCC_LSEConfig](#) (uint8_t RCC_LSE)
Configures the External Low Speed oscillator (LSE).
- void [RCC_LSICmd](#) (FunctionalState NewState)

- Enables or disables the Internal Low Speed oscillator (LSI).*

 - void [RCC_PLLConfig](#) (uint32_t RCC_PLLSource, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP, uint32_t PLLQ)

Configures the main PLL clock source, multiplication and division factors.

 - void [RCC_PLLCmd](#) (FunctionalState NewState)

Enables or disables the main PLL.

 - void [RCC_PLLI2SConfig](#) (uint32_t PLLI2SN, uint32_t PLLI2SR)

Configures the PLLI2S clock multiplication and division factors.

 - void [RCC_PLLI2SCmd](#) (FunctionalState NewState)

Enables or disables the PLLI2S.

 - void [RCC_ClockSecuritySystemCmd](#) (FunctionalState NewState)

Enables or disables the Clock Security System.

 - void [RCC_MCO1Config](#) (uint32_t RCC_MCO1Source, uint32_t RCC_MCO1Div)

Selects the clock source to output on MCO1 pin(PA8).

 - void [RCC_MCO2Config](#) (uint32_t RCC_MCO2Source, uint32_t RCC_MCO2Div)

Selects the clock source to output on MCO2 pin(PC9).

 - void [RCC_SYSClkConfig](#) (uint32_t RCC_SYSClkSource)

Configures the system clock (SYSClk).

 - uint8_t [RCC_GetSYSClkSource](#) (void)

Returns the clock source used as system clock.

 - void [RCC_HCLKConfig](#) (uint32_t RCC_SYSClk)

Configures the AHB clock (HCLK).

 - void [RCC_PCLK1Config](#) (uint32_t RCC_HCLK)

Configures the Low Speed APB clock (PCLK1).

 - void [RCC_PCLK2Config](#) (uint32_t RCC_HCLK)

Configures the High Speed APB clock (PCLK2).

 - void [RCC_GetClocksFreq](#) ([RCC_ClocksTypeDef](#) *RCC_Clocks)

Returns the frequencies of different on chip clocks; SYSClk, HCLK, PCLK1 and PCLK2.
 - void [RCC_RTCCLKConfig](#) (uint32_t RCC_RTCCLKSource)
- Configures the RTC clock (RTCCLK).*
- void [RCC_RTCCLKCmd](#) (FunctionalState NewState)
- Enables or disables the RTC clock.*
- void [RCC_BackupResetCmd](#) (FunctionalState NewState)
- Forces or releases the Backup domain reset.*
- void [RCC_I2SCLKConfig](#) (uint32_t RCC_I2SCLKSource)
- Configures the I2S clock source (I2SCLK).*
- void [RCC_AHB1PeriphClockCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
- Enables or disables the AHB1 peripheral clock.*
- void [RCC_AHB2PeriphClockCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
- Enables or disables the AHB2 peripheral clock.*
- void [RCC_AHB3PeriphClockCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
- Enables or disables the AHB3 peripheral clock.*
- void [RCC_APB1PeriphClockCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
- Enables or disables the Low Speed APB (APB1) peripheral clock.*
- void [RCC_APB2PeriphClockCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
- Enables or disables the High Speed APB (APB2) peripheral clock.*
- void [RCC_AHB1PeriphResetCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
- Forces or releases AHB1 peripheral reset.*
- void [RCC_AHB2PeriphResetCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
- Forces or releases AHB2 peripheral reset.*

- void [RCC_AHB3PeriphResetCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Forces or releases AHB3 peripheral reset.
- void [RCC_APB1PeriphResetCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Forces or releases Low Speed APB (APB1) peripheral reset.
- void [RCC_APB2PeriphResetCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Forces or releases High Speed APB (APB2) peripheral reset.
- void [RCC_AHB1PeriphClockLPModeCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
Enables or disables the AHB1 peripheral clock during Low Power (Sleep) mode.
- void [RCC_AHB2PeriphClockLPModeCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
Enables or disables the AHB2 peripheral clock during Low Power (Sleep) mode.
- void [RCC_AHB3PeriphClockLPModeCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Enables or disables the AHB3 peripheral clock during Low Power (Sleep) mode.
- void [RCC_APB1PeriphClockLPModeCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Enables or disables the APB1 peripheral clock during Low Power (Sleep) mode.
- void [RCC_APB2PeriphClockLPModeCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Enables or disables the APB2 peripheral clock during Low Power (Sleep) mode.
- void [RCC_ITConfig](#) (uint8_t RCC_IT, FunctionalState NewState)
Enables or disables the specified RCC interrupts.
- FlagStatus [RCC_GetFlagStatus](#) (uint8_t RCC_FLAG)
Checks whether the specified RCC flag is set or not.
- void [RCC_ClearFlag](#) (void)
Clears the RCC reset flags. The reset flags are: RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST, RCC_FLAG_LPWRRST.
- ITStatus [RCC_GetITStatus](#) (uint8_t RCC_IT)
Checks whether the specified RCC interrupt has occurred or not.
- void [RCC_ClearITPendingBit](#) (uint8_t RCC_IT)
Clears the RCC's interrupt pending bits.

4.1.5.1 Detailed Description

RCC driver modules.

4.1.5.2 Macro Definition Documentation

4.1.5.2.1 BDCR_ADDRESS

```
#define BDCR_ADDRESS (PERIPH_BASE + BDCR_OFFSET)
```

4.1.5.2.2 BDCR_BDRST_BB

```
#define BDCR_BDRST_BB (PERIPH_BB_BASE + (BDCR_OFFSET * 32) + (BDRST_BitNumber * 4))
```

4.1.5.2.3 BDCR_OFFSET

```
#define BDCR_OFFSET (RCC_OFFSET + 0x70)
```

4.1.5.2.4 BDCR_RTCEN_BB

```
#define BDCR_RTCEN_BB (PERIPH_BB_BASE + (BDCR_OFFSET * 32) + (RTCEN_BitNumber * 4))
```

4.1.5.2.5 BDRST_BitNumber

```
#define BDRST_BitNumber 0x10
```

4.1.5.2.6 CFGR_I2SSRC_BB

```
#define CFGR_I2SSRC_BB (PERIPH_BB_BASE + (CFGR_OFFSET * 32) + (I2SSRC_BitNumber * 4))
```

4.1.5.2.7 CFGR_MCO1_RESET_MASK

```
#define CFGR_MCO1_RESET_MASK ((uint32_t)0xF89FFFFFF)
```

4.1.5.2.8 CFGR_MCO2_RESET_MASK

```
#define CFGR_MCO2_RESET_MASK ((uint32_t)0x07FFFFFF)
```

4.1.5.2.9 CFGR_OFFSET

```
#define CFGR_OFFSET (RCC_OFFSET + 0x08)
```

4.1.5.2.10 CIR_BYTE2_ADDRESS

```
#define CIR_BYTE2_ADDRESS ((uint32_t)(RCC_BASE + 0x0C + 0x01))
```

4.1.5.2.11 CIR_BYTE3_ADDRESS

```
#define CIR_BYTE3_ADDRESS ((uint32_t)(RCC_BASE + 0x0C + 0x02))
```

4.1.5.2.12 CR_BYTE3_ADDRESS

```
#define CR_BYTE3_ADDRESS ((uint32_t)0x40023802)
```

4.1.5.2.13 CR_CSSON_BB

```
#define CR_CSSON_BB (PERIPH_BB_BASE + (CR_OFFSET * 32) + (CSSON_BitNumber * 4))
```

4.1.5.2.14 CR_HSION_BB

```
#define CR_HSION_BB (PERIPH_BB_BASE + (CR_OFFSET * 32) + (HSION_BitNumber * 4))
```

4.1.5.2.15 CR_OFFSET

```
#define CR_OFFSET (RCC_OFFSET + 0x00)
```

4.1.5.2.16 CR_PLLI2SON_BB

```
#define CR_PLLI2SON_BB (PERIPH_BB_BASE + (CR_OFFSET * 32) + (PLLI2SON_BitNumber * 4))
```

4.1.5.2.17 CR_PLLON_BB

```
#define CR_PLLON_BB (PERIPH_BB_BASE + (CR_OFFSET * 32) + (PLLON_BitNumber * 4))
```

4.1.5.2.18 CSR_LSION_BB

```
#define CSR_LSION_BB (PERIPH_BB_BASE + (CSR_OFFSET * 32) + (LSION_BitNumber * 4))
```

4.1.5.2.19 CSR_OFFSET

```
#define CSR_OFFSET (RCC_OFFSET + 0x74)
```

4.1.5.2.20 CSSON_BitNumber

```
#define CSSON_BitNumber 0x13
```

4.1.5.2.21 FLAG_MASK

```
#define FLAG_MASK ((uint8_t)0x1F)
```

4.1.5.2.22 HSION_BitNumber

```
#define HSION_BitNumber 0x00
```

4.1.5.2.23 I2SSRC_BitNumber

```
#define I2SSRC_BitNumber 0x17
```

4.1.5.2.24 LSION_BitNumber

```
#define LSION_BitNumber 0x00
```

4.1.5.2.25 PLLI2SON_BitNumber

```
#define PLLI2SON_BitNumber 0x1A
```

4.1.5.2.26 PLLON_BitNumber

```
#define PLLON_BitNumber 0x18
```

4.1.5.2.27 RCC_OFFSET

```
#define RCC_OFFSET (RCC_BASE - PERIPH_BASE)
```

4.1.5.2.28 RTCEN_BitNumber

```
#define RTCEN_BitNumber 0x0F
```

4.1.5.3 Function Documentation**4.1.5.3.1 RCC_AdjustHSICalibrationValue()**

```
void RCC_AdjustHSICalibrationValue (
    uint8_t HSICalibrationValue )
```

Adjusts the Internal High Speed oscillator (HSI) calibration value.

Note

The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

Parameters

<i>HSICalibrationValue</i>	specifies the calibration trimming value. This parameter must be a number between 0 and 0x1F.
----------------------------	---

Return values

<i>None</i>	
-------------	--

4.1.5.3.2 RCC_AHB1PeriphClockCmd()

```
void RCC_AHB1PeriphClockCmd (
    uint32_t RCC_AHB1Periph,
```

```
FunctionalState NewState )
```

Enables or disables the AHB1 peripheral clock.

Note

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

Parameters

<i>RCC_AHBPeriph</i>	<p>specifies the AHB1 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_AHB1Periph_GPIOA: GPIOA clock • RCC_AHB1Periph_GPIOB: GPIOB clock • RCC_AHB1Periph_GPIOC: GPIOC clock • RCC_AHB1Periph_GPIOD: GPIOD clock • RCC_AHB1Periph_GPIOE: GPIOE clock • RCC_AHB1Periph_GPIOF: GPIOF clock • RCC_AHB1Periph_GPIOG: GPIOG clock • RCC_AHB1Periph_GPIOG: GPIOG clock • RCC_AHB1Periph_GPIOI: GPIOI clock • RCC_AHB1Periph_CRC: CRC clock • RCC_AHB1Periph_BKPSRAM: BKPSRAM interface clock • RCC_AHB1Periph_CCMDATARAMEN CCM data RAM interface clock • RCC_AHB1Periph_DMA1: DMA1 clock • RCC_AHB1Periph_DMA2: DMA2 clock • RCC_AHB1Periph_ETH_MAC: Ethernet MAC clock • RCC_AHB1Periph_ETH_MAC_Tx: Ethernet Transmission clock • RCC_AHB1Periph_ETH_MAC_Rx: Ethernet Reception clock • RCC_AHB1Periph_ETH_MAC_PTP: Ethernet PTP clock • RCC_AHB1Periph_OTG_HS: USB OTG HS clock • RCC_AHB1Periph_OTG_HS_ULPI: USB OTG HS ULPI clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.3 RCC_AHB1PeriphClockLPModeCmd()

```
void RCC_AHB1PeriphClockLPModeCmd (
```

```
uint32_t RCC_AHB1Periph,
FunctionalState NewState )
```

Enables or disables the AHB1 peripheral clock during Low Power (Sleep) mode.

Note

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again.

By default, all peripheral clocks are enabled during SLEEP mode.

Parameters

<i>RCC_AHBPeriph</i>	<p>specifies the AHB1 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_AHB1Periph_GPIOA: GPIOA clock • RCC_AHB1Periph_GPIOB: GPIOB clock • RCC_AHB1Periph_GPIOC: GPIOC clock • RCC_AHB1Periph_GPIOD: GPIOD clock • RCC_AHB1Periph_GPIOE: GPIOE clock • RCC_AHB1Periph_GPIOF: GPIOF clock • RCC_AHB1Periph_GPIOG: GPIOG clock • RCC_AHB1Periph_GPIOG: GPIOG clock • RCC_AHB1Periph_GPIOI: GPIOI clock • RCC_AHB1Periph_CRC: CRC clock • RCC_AHB1Periph_BKPSRAM: BKPSRAM interface clock • RCC_AHB1Periph_DMA1: DMA1 clock • RCC_AHB1Periph_DMA2: DMA2 clock • RCC_AHB1Periph_ETH_MAC: Ethernet MAC clock • RCC_AHB1Periph_ETH_MAC_Tx: Ethernet Transmission clock • RCC_AHB1Periph_ETH_MAC_Rx: Ethernet Reception clock • RCC_AHB1Periph_ETH_MAC_PTP: Ethernet PTP clock • RCC_AHB1Periph_OTG_HS: USB OTG HS clock • RCC_AHB1Periph_OTG_HS_ULPI: USB OTG HS ULPI clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.4 RCC_AHB1PeriphResetCmd()

```
void RCC_AHB1PeriphResetCmd (
```



```
uint32_t RCC_AHB1Periph,
FunctionalState NewState )
```

Forces or releases AHB1 peripheral reset.

Parameters

<i>RCC_AHB1Periph</i>	<p>specifies the AHB1 peripheral to reset. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_AHB1Periph_GPIOA: GPIOA clock • RCC_AHB1Periph_GPIOB: GPIOB clock • RCC_AHB1Periph_GPIOC: GPIOC clock • RCC_AHB1Periph_GPIOD: GPIOD clock • RCC_AHB1Periph_GPIOE: GPIOE clock • RCC_AHB1Periph_GPIOF: GPIOF clock • RCC_AHB1Periph_GPIOG: GPIOG clock • RCC_AHB1Periph_GPIOG: GPIOG clock • RCC_AHB1Periph_GPIOI: GPIOI clock • RCC_AHB1Periph_CRC: CRC clock • RCC_AHB1Periph_DMA1: DMA1 clock • RCC_AHB1Periph_DMA2: DMA2 clock • RCC_AHB1Periph_ETH_MAC: Ethernet MAC clock • RCC_AHB1Periph_OTG_HS: USB OTG HS clock
<i>NewState</i>	new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.5 RCC_AHB2PeriphClockCmd()

```
void RCC_AHB2PeriphClockCmd (
uint32_t RCC_AHB2Periph,
FunctionalState NewState )
```

Enables or disables the AHB2 peripheral clock.

Note

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

Parameters

<i>RCC_AHBPeriph</i>	specifies the AHB2 peripheral to gates its clock. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • RCC_AHB2Periph_DCMI: DCMI clock • RCC_AHB2Periph_Cryp: CRYp clock • RCC_AHB2Periph_HASH: HASH clock • RCC_AHB2Periph_RNG: RNG clock • RCC_AHB2Periph_OTG_FS: USB OTG FS clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.6 RCC_AHB2PeriphClockLPModeCmd()

```
void RCC_AHB2PeriphClockLPModeCmd (
    uint32_t RCC_AHB2Periph,
    FunctionalState NewState )
```

Enables or disables the AHB2 peripheral clock during Low Power (Sleep) mode.

Note

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again.

By default, all peripheral clocks are enabled during SLEEP mode.

Parameters

<i>RCC_AHBPeriph</i>	specifies the AHB2 peripheral to gates its clock. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • RCC_AHB2Periph_DCMI: DCMI clock • RCC_AHB2Periph_Cryp: CRYp clock • RCC_AHB2Periph_HASH: HASH clock • RCC_AHB2Periph_RNG: RNG clock • RCC_AHB2Periph_OTG_FS: USB OTG FS clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.7 RCC_AHB2PeriphResetCmd()

```
void RCC_AHB2PeriphResetCmd (
    uint32_t RCC_AHB2Periph,
    FunctionalState NewState )
```

Forces or releases AHB2 peripheral reset.

Parameters

<i>RCC_AHB2Periph</i>	specifies the AHB2 peripheral to reset. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • RCC_AHB2Periph_DCMI: DCMI clock • RCC_AHB2Periph_Cryp: CRYP clock • RCC_AHB2Periph_HASH: HASH clock • RCC_AHB2Periph_RNG: RNG clock • RCC_AHB2Periph_OTG_FS: USB OTG FS clock
<i>NewState</i>	new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.8 RCC_AHB3PeriphClockCmd()

```
void RCC_AHB3PeriphClockCmd (
    uint32_t RCC_AHB3Periph,
    FunctionalState NewState )
```

Enables or disables the AHB3 peripheral clock.

Note

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

Parameters

<i>RCC_AHBPeriph</i>	specifies the AHB3 peripheral to gates its clock. This parameter must be: RCC_AHB3Periph_FSMC
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.9 RCC_AHB3PeriphClockLPModeCmd()

```
void RCC_AHB3PeriphClockLPModeCmd (
    uint32_t RCC_AHB3Periph,
    FunctionalState NewState )
```

Enables or disables the AHB3 peripheral clock during Low Power (Sleep) mode.

Note

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again.

By default, all peripheral clocks are enabled during SLEEP mode.

Parameters

<i>RCC_AHBPeriph</i>	specifies the AHB3 peripheral to gates its clock. This parameter must be: RCC_AHB3Periph_FSMC
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.10 RCC_AHB3PeriphResetCmd()

```
void RCC_AHB3PeriphResetCmd (
    uint32_t RCC_AHB3Periph,
    FunctionalState NewState )
```

Forces or releases AHB3 peripheral reset.

Parameters

<i>RCC_AHB3Periph</i>	specifies the AHB3 peripheral to reset. This parameter must be: RCC_AHB3Periph_FSMC
<i>NewState</i>	new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.11 RCC_APB1PeriphClockCmd()

```
void RCC_APB1PeriphClockCmd (
    uint32_t RCC_APB1Periph,
    FunctionalState NewState )
```

Enables or disables the Low Speed APB (APB1) peripheral clock.

Note

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

Parameters

<i>RCC_APB1Periph</i>	<p>specifies the APB1 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_APB1Periph_TIM2: TIM2 clock • RCC_APB1Periph_TIM3: TIM3 clock • RCC_APB1Periph_TIM4: TIM4 clock • RCC_APB1Periph_TIM5: TIM5 clock • RCC_APB1Periph_TIM6: TIM6 clock • RCC_APB1Periph_TIM7: TIM7 clock • RCC_APB1Periph_TIM12: TIM12 clock • RCC_APB1Periph_TIM13: TIM13 clock • RCC_APB1Periph_TIM14: TIM14 clock • RCC_APB1Periph_WWDG: WWDG clock • RCC_APB1Periph_SPI2: SPI2 clock • RCC_APB1Periph_SPI3: SPI3 clock • RCC_APB1Periph_USART2: USART2 clock • RCC_APB1Periph_USART3: USART3 clock • RCC_APB1Periph_UART4: UART4 clock • RCC_APB1Periph_UART5: UART5 clock • RCC_APB1Periph_I2C1: I2C1 clock • RCC_APB1Periph_I2C2: I2C2 clock • RCC_APB1Periph_I2C3: I2C3 clock • RCC_APB1Periph_CAN1: CAN1 clock • RCC_APB1Periph_CAN2: CAN2 clock • RCC_APB1Periph_PWR: PWR clock • RCC_APB1Periph_DAC: DAC clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.12 RCC_APB1PeriphClockLPModeCmd()

```
void RCC_APB1PeriphClockLPModeCmd (
    uint32_t RCC_APB1Periph,
    FunctionalState NewState )
```

Enables or disables the APB1 peripheral clock during Low Power (Sleep) mode.

Note

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again.

By default, all peripheral clocks are enabled during SLEEP mode.

Parameters

<i>RCC_APB1Periph</i>	<p>specifies the APB1 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_APB1Periph_TIM2: TIM2 clock • RCC_APB1Periph_TIM3: TIM3 clock • RCC_APB1Periph_TIM4: TIM4 clock • RCC_APB1Periph_TIM5: TIM5 clock • RCC_APB1Periph_TIM6: TIM6 clock • RCC_APB1Periph_TIM7: TIM7 clock • RCC_APB1Periph_TIM12: TIM12 clock • RCC_APB1Periph_TIM13: TIM13 clock • RCC_APB1Periph_TIM14: TIM14 clock • RCC_APB1Periph_WWDG: WWDG clock • RCC_APB1Periph_SPI2: SPI2 clock • RCC_APB1Periph_SPI3: SPI3 clock • RCC_APB1Periph_USART2: USART2 clock • RCC_APB1Periph_USART3: USART3 clock • RCC_APB1Periph_UART4: UART4 clock • RCC_APB1Periph_UART5: UART5 clock • RCC_APB1Periph_I2C1: I2C1 clock • RCC_APB1Periph_I2C2: I2C2 clock • RCC_APB1Periph_I2C3: I2C3 clock • RCC_APB1Periph_CAN1: CAN1 clock • RCC_APB1Periph_CAN2: CAN2 clock • RCC_APB1Periph_PWR: PWR clock • RCC_APB1Periph_DAC: DAC clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.13 RCC_APB1PeriphResetCmd()

```
void RCC_APB1PeriphResetCmd (
    uint32_t RCC_APB1Periph,
    FunctionalState NewState )
```

Forces or releases Low Speed APB (APB1) peripheral reset.

Parameters

<i>RCC_APB1Periph</i>	<p>specifies the APB1 peripheral to reset. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_APB1Periph_TIM2: TIM2 clock • RCC_APB1Periph_TIM3: TIM3 clock • RCC_APB1Periph_TIM4: TIM4 clock • RCC_APB1Periph_TIM5: TIM5 clock • RCC_APB1Periph_TIM6: TIM6 clock • RCC_APB1Periph_TIM7: TIM7 clock • RCC_APB1Periph_TIM12: TIM12 clock • RCC_APB1Periph_TIM13: TIM13 clock • RCC_APB1Periph_TIM14: TIM14 clock • RCC_APB1Periph_WWDG: WWDG clock • RCC_APB1Periph_SPI2: SPI2 clock • RCC_APB1Periph_SPI3: SPI3 clock • RCC_APB1Periph_USART2: USART2 clock • RCC_APB1Periph_USART3: USART3 clock • RCC_APB1Periph_UART4: UART4 clock • RCC_APB1Periph_UART5: UART5 clock • RCC_APB1Periph_I2C1: I2C1 clock • RCC_APB1Periph_I2C2: I2C2 clock • RCC_APB1Periph_I2C3: I2C3 clock • RCC_APB1Periph_CAN1: CAN1 clock • RCC_APB1Periph_CAN2: CAN2 clock • RCC_APB1Periph_PWR: PWR clock • RCC_APB1Periph_DAC: DAC clock
<i>NewState</i>	new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.14 RCC_APB2PeriphClockCmd()

```
void RCC_APB2PeriphClockCmd (
    uint32_t RCC_APB2Periph,
    FunctionalState NewState )
```

Enables or disables the High Speed APB (APB2) peripheral clock.

Note

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

Parameters

<i>RCC_APB2Periph</i>	<p>specifies the APB2 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_APB2Periph_TIM1: TIM1 clock • RCC_APB2Periph_TIM8: TIM8 clock • RCC_APB2Periph_USART1: USART1 clock • RCC_APB2Periph_USART6: USART6 clock • RCC_APB2Periph_ADC1: ADC1 clock • RCC_APB2Periph_ADC2: ADC2 clock • RCC_APB2Periph_ADC3: ADC3 clock • RCC_APB2Periph_SDIO: SDIO clock • RCC_APB2Periph_SPI1: SPI1 clock • RCC_APB2Periph_SYSCFG: SYSCFG clock • RCC_APB2Periph_TIM9: TIM9 clock • RCC_APB2Periph_TIM10: TIM10 clock • RCC_APB2Periph_TIM11: TIM11 clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.3.15 RCC_APB2PeriphClockLPModeCmd()

```
void RCC_APB2PeriphClockLPModeCmd (
    uint32_t RCC_APB2Periph,
    FunctionalState NewState )
```

Enables or disables the APB2 peripheral clock during Low Power (Sleep) mode.

Note

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again.

By default, all peripheral clocks are enabled during SLEEP mode.

Parameters

<i>RCC_APB2Periph</i>	<p>specifies the APB2 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_APB2Periph_TIM1</code>: TIM1 clock • <code>RCC_APB2Periph_TIM8</code>: TIM8 clock • <code>RCC_APB2Periph_USART1</code>: USART1 clock • <code>RCC_APB2Periph_USART6</code>: USART6 clock • <code>RCC_APB2Periph_ADC1</code>: ADC1 clock • <code>RCC_APB2Periph_ADC2</code>: ADC2 clock • <code>RCC_APB2Periph_ADC3</code>: ADC3 clock • <code>RCC_APB2Periph_SDIO</code>: SDIO clock • <code>RCC_APB2Periph_SPI1</code>: SPI1 clock • <code>RCC_APB2Periph_SYSCFG</code>: SYSCFG clock • <code>RCC_APB2Periph_TIM9</code>: TIM9 clock • <code>RCC_APB2Periph_TIM10</code>: TIM10 clock • <code>RCC_APB2Periph_TIM11</code>: TIM11 clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .

Return values

<i>None</i>	
-------------	--

4.1.5.3.16 RCC_APB2PeriphResetCmd()

```
void RCC_APB2PeriphResetCmd (
    uint32_t RCC_APB2Periph,
    FunctionalState NewState )
```

Forces or releases High Speed APB (APB2) peripheral reset.

Parameters

<i>RCC_APB2Periph</i>	<p>specifies the APB2 peripheral to reset. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_APB2Periph_TIM1</code>: TIM1 clock • <code>RCC_APB2Periph_TIM8</code>: TIM8 clock • <code>RCC_APB2Periph_USART1</code>: USART1 clock • <code>RCC_APB2Periph_USART6</code>: USART6 clock • <code>RCC_APB2Periph_ADC1</code>: ADC1 clock • <code>RCC_APB2Periph_ADC2</code>: ADC2 clock • <code>RCC_APB2Periph_ADC3</code>: ADC3 clock • <code>RCC_APB2Periph_SDIO</code>: SDIO clock • <code>RCC_APB2Periph_SPI1</code>: SPI1 clock • <code>RCC_APB2Periph_SYSCFG</code>: SYSCFG clock • <code>RCC_APB2Periph_TIM9</code>: TIM9 clock • <code>RCC_APB2Periph_TIM10</code>: TIM10 clock • <code>RCC_APB2Periph_TIM11</code>: TIM11 clock
<i>NewState</i>	new state of the specified peripheral reset. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .

Return values

<i>None</i>	
-------------	--

4.1.5.3.17 RCC_BackupResetCmd()

```
void RCC_BackupResetCmd (
    FunctionalState NewState )
```

Forces or releases the Backup domain reset.

Note

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.

Parameters

<i>NewState</i>	new state of the Backup domain reset. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .
-----------------	--

Return values

<i>None</i>	
-------------	--

4.1.5.3.18 RCC_ClearFlag()

```
void RCC_ClearFlag (
    void )
```

Clears the RCC reset flags. The reset flags are: RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST, RCC_FLAG_LPWRST.

Parameters

None	
------	--

Return values

None	
------	--

4.1.5.3.19 RCC_ClearITPendingBit()

```
void RCC_ClearITPendingBit (
    uint8_t RCC_IT )
```

Clears the RCC's interrupt pending bits.

Parameters

<i>RCC_IT</i>	<p>specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_IT_LSIRDY: LSI ready interrupt • RCC_IT_LSERDY: LSE ready interrupt • RCC_IT_HSIRDY: HSI ready interrupt • RCC_IT_HSERDY: HSE ready interrupt • RCC_IT_PLLRDY: main PLL ready interrupt • RCC_IT_PLLI2SRDY: PLLI2S ready interrupt • RCC_IT_CSS: Clock Security System interrupt
---------------	--

Return values

None	
------	--

4.1.5.3.20 RCC_ClockSecuritySystemCmd()

```
void RCC_ClockSecuritySystemCmd (
    FunctionalState NewState )
```

Enables or disables the Clock Security System.

Note

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-M4 NMI (Non-Maskable Interrupt) exception vector.

Parameters

<i>NewState</i>	new state of the Clock Security System. This parameter can be: ENABLE or DISABLE.
-----------------	---

Return values

<i>None</i>	
-------------	--

4.1.5.3.21 RCC_DeInit()

```
void RCC_DeInit (
    void )
```

Resets the RCC clock configuration to the default reset state.

Note

The default reset state of the clock configuration is given below:

- HSI ON and used as system clock source
- HSE, PLL and PLLI2S OFF
- AHB, APB1 and APB2 prescaler set to 1.
- CSS, MCO1 and MCO2 OFF
- All interrupts disabled

This function doesn't modify the configuration of the

- Peripheral clocks
- LSI, LSE and RTC clocks

Parameters

<i>None</i>	
-------------	--

Return values

<i>None</i>	
-------------	--

4.1.5.3.22 RCC_GetClocksFreq()

```
void RCC_GetClocksFreq (
    RCC_ClocksTypeDef * RCC_Clocks )
```

Returns the frequencies of different on chip clocks; SYSCLK, HCLK, PCLK1 and PCLK2.

Note

The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:

If SYSCLK source is HSI, function returns values based on [HSI_VALUE\(*\)](#)

If SYSCLK source is HSE, function returns values based on [HSE_VALUE\(**\)](#)

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 stm32f4xx.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 stm32f4xx.h file (default value 25 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.

Parameters

<i>RCC_Clocks</i>	pointer to a RCC_ClocksTypeDef structure which will hold the clocks frequencies.
-------------------	--

Note

This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.

Each time SYSCLK, HCLK, PCLK1 and/or PCLK2 clock changes, this function must be called to update the structure's field. Otherwise, any configuration based on this function will be incorrect.

Return values

<i>None</i>	
-------------	--

4.1.5.3.23 RCC_GetFlagStatus()

```
FlagStatus RCC_GetFlagStatus (
    uint8_t RCC_FLAG )
```

Checks whether the specified RCC flag is set or not.

Parameters

<i>RCC_FLAG</i>	<p>specifies the flag to check. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • RCC_FLAG_HSIRDY: HSI oscillator clock ready • RCC_FLAG_HSERDY: HSE oscillator clock ready • RCC_FLAG_PLLRDY: main PLL clock ready • RCC_FLAG_PLLI2SRDY: PLLI2S clock ready • RCC_FLAG_LSERDY: LSE oscillator clock ready • RCC_FLAG_LSIRDY: LSI oscillator clock ready • RCC_FLAG_BORRST: POR/PDR or BOR 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 values

<i>The</i>	new state of RCC_FLAG (SET or RESET).
------------	---------------------------------------

4.1.5.3.24 RCC_GetITStatus()

```
ITStatus RCC_GetITStatus (
    uint8_t RCC_IT )
```

Checks whether the specified RCC interrupt has occurred or not.

Parameters

<i>RCC_IT</i>	<p>specifies the RCC interrupt source to check. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • RCC_IT_LSIRDY: LSI ready interrupt • RCC_IT_LSERDY: LSE ready interrupt • RCC_IT_HSIIRDY: HSI ready interrupt • RCC_IT_HSERDY: HSE ready interrupt • RCC_IT_PLLRDY: main PLL ready interrupt • RCC_IT_PLLI2SRDY: PLLI2S ready interrupt • RCC_IT_CSS: Clock Security System interrupt
---------------	--

Return values

<i>The</i>	new state of RCC_IT (SET or RESET).
------------	-------------------------------------

4.1.5.3.25 RCC_GetSYSCLKSource()

```
uint8_t RCC_GetSYSCLKSource (
    void )
```

Returns the clock source used as system clock.

Parameters

<i>None</i>	
-------------	--

Return values

<i>The</i>	<p>clock source used as system clock. The returned value can be one of the following:</p> <ul style="list-style-type: none"> • 0x00: HSI used as system clock • 0x04: HSE used as system clock • 0x08: PLL used as system clock
------------	--

4.1.5.3.26 RCC_HCLKConfig()

```
void RCC_HCLKConfig (
    uint32_t RCC_SYSCLK )
```

Configures the AHB clock (HCLK).

Note

Depending on the device voltage range, the software has to set correctly these bits to ensure that HCLK not exceed the maximum allowed frequency (for more details refer to section above "CPU, AHB and APB busses clocks configuration functions")

Parameters

<i>RCC_SYSCLK</i>	<p>defines the AHB clock divider. This clock is derived from the system clock (SYSCLK). This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • RCC_SYSCLK_Div1: AHB clock = SYSCLK • RCC_SYSCLK_Div2: AHB clock = SYSCLK/2 • RCC_SYSCLK_Div4: AHB clock = SYSCLK/4 • RCC_SYSCLK_Div8: AHB clock = SYSCLK/8 • RCC_SYSCLK_Div16: AHB clock = SYSCLK/16 • RCC_SYSCLK_Div64: AHB clock = SYSCLK/64 • RCC_SYSCLK_Div128: AHB clock = SYSCLK/128 • RCC_SYSCLK_Div256: AHB clock = SYSCLK/256 • RCC_SYSCLK_Div512: AHB clock = SYSCLK/512
-------------------	--

Return values

<i>None</i>	
-------------	--

4.1.5.3.27 RCC_HSEConfig()

```
void RCC_HSEConfig (
    uint8_t RCC_HSE )
```

Configures the External High Speed oscillator (HSE).

Note

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.

Parameters

<i>RCC_HSE</i>	specifies the new state of the HSE. This parameter can be one of the following values: <ul style="list-style-type: none"> • <i>RCC_HSE_OFF</i>: turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles. • <i>RCC_HSE_ON</i>: turn ON the HSE oscillator • <i>RCC_HSE_Bypass</i>: HSE oscillator bypassed with external clock
----------------	--

Return values

<i>None</i>	
-------------	--

4.1.5.3.28 RCC_HSICmd()

```
void RCC_HSICmd (
    FunctionalState NewState )
```

Enables or disables the Internal High Speed oscillator (HSI).

Note

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.

Parameters

<i>NewState</i>	new state of the HSI. This parameter can be: ENABLE or DISABLE.
-----------------	---

Note

When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

Return values

<i>None</i>	
-------------	--

4.1.5.3.29 RCC_I2SCLKConfig()

```
void RCC_I2SCLKConfig (
    uint32_t RCC_I2SCLKSource )
```

Configures the I2S clock source (I2SCLK).

Note

This function must be called before enabling the I2S APB clock.

Parameters

<i>RCC_I2SCLKSource</i>	specifies the I2S clock source. This parameter can be one of the following values: <ul style="list-style-type: none"> • <code>RCC_I2S2CLKSource_PLLI2S</code>: PLLI2S clock used as I2S clock source • <code>RCC_I2S2CLKSource_Ext</code>: External clock mapped on the I2S_CKIN pin used as I2S clock source
-------------------------	---

Return values

<i>None</i>	
-------------	--

4.1.5.3.30 RCC_ITConfig()

```
void RCC_ITConfig (
    uint8_t RCC_IT,
    FunctionalState NewState )
```

Enables or disables the specified RCC interrupts.

Parameters

<i>RCC_IT</i>	specifies the RCC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • <code>RCC_IT_LSIRDY</code>: LSI ready interrupt • <code>RCC_IT_LSERDY</code>: LSE ready interrupt • <code>RCC_IT_HSIRDY</code>: HSI ready interrupt • <code>RCC_IT_HSERDY</code>: HSE ready interrupt • <code>RCC_IT_PLLRDY</code>: main PLL ready interrupt • <code>RCC_IT_PLLI2SRDY</code>: PLLI2S ready interrupt
<i>NewState</i>	new state of the specified RCC interrupts. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .

Return values

<i>None</i>	
-------------	--

4.1.5.3.31 RCC_LSEConfig()

```
void RCC_LSEConfig (
    uint8_t RCC_LSE )
```

Configures the External Low Speed oscillator (LSE).

Note

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 `PWR_BackupAccessCmd(ENABLE)` 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.

Parameters

<i>RCC_LSE</i>	specifies the new state of the LSE. This parameter can be one of the following values: <ul style="list-style-type: none"> • <code>RCC_LSE_OFF</code>: turn OFF the LSE oscillator, <code>LSERDY</code> flag goes low after 6 LSE oscillator clock cycles. • <code>RCC_LSE_ON</code>: turn ON the LSE oscillator • <code>RCC_LSE_Bypass</code>: LSE oscillator bypassed with external clock
----------------	---

Return values

<i>None</i>	
-------------	--

4.1.5.3.32 RCC_LSICmd()

```
void RCC_LSICmd (
    FunctionalState NewState )
```

Enables or disables the Internal Low Speed oscillator (LSI).

Note

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.

Parameters

<i>NewState</i>	new state of the LSI. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .
-----------------	--

Note

When the LSI is stopped, `LSIRDY` flag goes low after 6 LSI oscillator clock cycles.

Return values

<i>None</i>	
-------------	--

4.1.5.3.33 RCC_MCO1Config()

```
void RCC_MCO1Config (
    uint32_t RCC_MCO1Source,
    uint32_t RCC_MCO1Div )
```

Selects the clock source to output on MCO1 pin(PA8).

Note

PA8 should be configured in alternate function mode.

Parameters

<i>RCC_MCO1Source</i>	specifies the clock source to output. This parameter can be one of the following values: <ul style="list-style-type: none"> • <i>RCC_MCO1Source_HSI</i>: HSI clock selected as MCO1 source • <i>RCC_MCO1Source_LSE</i>: LSE clock selected as MCO1 source • <i>RCC_MCO1Source_HSE</i>: HSE clock selected as MCO1 source • <i>RCC_MCO1Source_PLLCLK</i>: main PLL clock selected as MCO1 source
<i>RCC_MCO1Div</i>	specifies the MCO1 prescaler. This parameter can be one of the following values: <ul style="list-style-type: none"> • <i>RCC_MCO1Div_1</i>: no division applied to MCO1 clock • <i>RCC_MCO1Div_2</i>: division by 2 applied to MCO1 clock • <i>RCC_MCO1Div_3</i>: division by 3 applied to MCO1 clock • <i>RCC_MCO1Div_4</i>: division by 4 applied to MCO1 clock • <i>RCC_MCO1Div_5</i>: division by 5 applied to MCO1 clock

Return values

<i>None</i>	
-------------	--

4.1.5.3.34 RCC_MCO2Config()

```
void RCC_MCO2Config (
    uint32_t RCC_MCO2Source,
    uint32_t RCC_MCO2Div )
```

Selects the clock source to output on MCO2 pin(PC9).

Note

PC9 should be configured in alternate function mode.

Parameters

<i>RCC_MCO2Source</i>	specifies the clock source to output. This parameter can be one of the following values: <ul style="list-style-type: none"> • <i>RCC_MCO2Source_SYSCLK</i>: System clock (SYSCLK) selected as MCO2 source • <i>RCC_MCO2Source_PLLI2SCLK</i>: PLLI2S clock selected as MCO2 source • <i>RCC_MCO2Source_HSE</i>: HSE clock selected as MCO2 source • <i>RCC_MCO2Source_PLLCLK</i>: main PLL clock selected as MCO2 source
-----------------------	---

Parameters

<i>RCC_MCO2Div</i>	specifies the MCO2 prescaler. This parameter can be one of the following values: <ul style="list-style-type: none"> • <i>RCC_MCO2Div_1</i>: no division applied to MCO2 clock • <i>RCC_MCO2Div_2</i>: division by 2 applied to MCO2 clock • <i>RCC_MCO2Div_3</i>: division by 3 applied to MCO2 clock • <i>RCC_MCO2Div_4</i>: division by 4 applied to MCO2 clock • <i>RCC_MCO2Div_5</i>: division by 5 applied to MCO2 clock
--------------------	--

Return values

<i>None</i>	
-------------	--

4.1.5.3.35 RCC_PCLK1Config()

```
void RCC_PCLK1Config (
    uint32_t RCC_HCLK )
```

Configures the Low Speed APB clock (PCLK1).

Parameters

<i>RCC_HCLK</i>	defines the APB1 clock divider. This clock is derived from the AHB clock (HCLK). This parameter can be one of the following values: <ul style="list-style-type: none"> • <i>RCC_HCLK_Div1</i>: APB1 clock = HCLK • <i>RCC_HCLK_Div2</i>: APB1 clock = HCLK/2 • <i>RCC_HCLK_Div4</i>: APB1 clock = HCLK/4 • <i>RCC_HCLK_Div8</i>: APB1 clock = HCLK/8 • <i>RCC_HCLK_Div16</i>: APB1 clock = HCLK/16
-----------------	---

Return values

<i>None</i>	
-------------	--

4.1.5.3.36 RCC_PCLK2Config()

```
void RCC_PCLK2Config (
    uint32_t RCC_HCLK )
```

Configures the High Speed APB clock (PCLK2).

Parameters

<i>RCC_HCLK</i>	<p>defines the APB2 clock divider. This clock is derived from the AHB clock (HCLK). This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <i>RCC_HCLK_Div1</i>: APB2 clock = HCLK • <i>RCC_HCLK_Div2</i>: APB2 clock = HCLK/2 • <i>RCC_HCLK_Div4</i>: APB2 clock = HCLK/4 • <i>RCC_HCLK_Div8</i>: APB2 clock = HCLK/8 • <i>RCC_HCLK_Div16</i>: APB2 clock = HCLK/16
-----------------	--

Return values

<i>None</i>	
-------------	--

4.1.5.3.37 RCC_PLLCmd()

```
void RCC_PLLCmd (
    FunctionalState NewState )
```

Enables or disables the main PLL.

Note

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.

Parameters

<i>NewState</i>	new state of the main PLL. This parameter can be: ENABLE or DISABLE.
-----------------	--

Return values

<i>None</i>	
-------------	--

4.1.5.3.38 RCC_PLLConfig()

```
void RCC_PLLConfig (
    uint32_t RCC_PLLSource,
    uint32_t PLLM,
    uint32_t PLLN,
    uint32_t PLLP,
    uint32_t PLLQ )
```

Configures the main PLL clock source, multiplication and division factors.

Note

This function must be used only when the main PLL is disabled.

Parameters

<i>RCC_PLLSource</i>	specifies the PLL entry clock source. This parameter can be one of the following values: <ul style="list-style-type: none"> • <code>RCC_PLLSource_HSI</code>: HSI oscillator clock selected as PLL clock entry • <code>RCC_PLLSource_HSE</code>: HSE oscillator clock selected as PLL clock entry
----------------------	---

Note

This clock source (`RCC_PLLSource`) is common for the main PLL and PLLI2S.

Parameters

<i>PLLM</i>	specifies the division factor for PLL VCO input clock This parameter must be a number between 0 and 63.
-------------	---

Note

You have to set the `PLLM` parameter correctly to ensure that the VCO input frequency ranges from 1 to 2 MHz. It is recommended to select a frequency of 2 MHz to limit PLL jitter.

Parameters

<i>PLLN</i>	specifies the multiplication factor for PLL VCO output clock This parameter must be a number between 192 and 432.
-------------	---

Note

You have to set the `PLLN` parameter correctly to ensure that the VCO output frequency is between 192 and 432 MHz.

Parameters

<i>PLLp</i>	specifies the division factor for main system clock (SYSCLK) This parameter must be a number in the range {2, 4, 6, or 8}.
-------------	--

Note

You have to set the `PLLp` parameter correctly to not exceed 168 MHz on the System clock frequency.

Parameters

<i>PLLQ</i>	specifies the division factor for OTG FS, SDIO and RNG clocks This parameter must be a number between 4 and 15.
-------------	---

Note

If the USB OTG FS is used in your application, you have to set the `PLLQ` parameter correctly to have 48 MHz clock for the USB. However, the SDIO and RNG need a frequency lower than or equal to 48 MHz to work correctly.

Return values

<i>None</i>	
-------------	--

4.1.5.3.39 RCC_PLLI2SCmd()

```
void RCC_PLLI2SCmd (
    FunctionalState NewState )
```

Enables or disables the PLLI2S.

Note

The PLLI2S is disabled by hardware when entering STOP and STANDBY modes.

Parameters

<i>NewState</i>	new state of the PLLI2S. This parameter can be: ENABLE or DISABLE.
-----------------	--

Return values

<i>None</i>	
-------------	--

4.1.5.3.40 RCC_PLLI2SConfig()

```
void RCC_PLLI2SConfig (
    uint32_t PLLI2SN,
    uint32_t PLLI2SR )
```

Configures the PLLI2S clock multiplication and division factors.

Note

This function must be used only when the PLLI2S is disabled.

PLLI2S clock source is common with the main PLL (configured in RCC_PLLConfig function)

Parameters

<i>PLLI2SN</i>	specifies the multiplication factor for PLLI2S VCO output clock This parameter must be a number between 192 and 432.
----------------	--

Note

You have to set the PLLI2SN parameter correctly to ensure that the VCO output frequency is between 192 and 432 MHz.

Parameters

<i>PLLI2SR</i>	specifies the division factor for I2S clock This parameter must be a number between 2 and 7.
----------------	--

Note

You have to set the PLLI2SR parameter correctly to not exceed 192 MHz on the I2S clock frequency.

Return values

<i>None</i>	
-------------	--

4.1.5.3.41 RCC_RTCCLKCmd()

```
void RCC_RTCCLKCmd (
    FunctionalState NewState )
```

Enables or disables the RTC clock.

Note

This function must be used only after the RTC clock source was selected using the `RCC_RTCCLKConfig` function.

Parameters

<i>NewState</i>	new state of the RTC clock. This parameter can be: ENABLE or DISABLE.
-----------------	---

Return values

<i>None</i>	
-------------	--

4.1.5.3.42 RCC_RTCCLKConfig()

```
void RCC_RTCCLKConfig (
    uint32_t RCC_RTCCLKSource )
```

Configures the RTC clock (RTCCLK).

Note

As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using `PWR_BackupAccessCmd(ENABLE)` function before to configure the RTC clock source (to be done once after reset).

Once the RTC clock is configured it can't be changed unless the Backup domain is reset using [RCC_BackupResetCmd\(\)](#) function, or by a Power On Reset (POR).

Parameters

<i>RCC_RTCCLKSource</i>	<p>specifies the RTC clock source. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_RTCCLKSource_LSE</code>: LSE selected as RTC clock • <code>RCC_RTCCLKSource_LSI</code>: LSI selected as RTC clock • <code>RCC_RTCCLKSource_HSE_Divx</code>: HSE clock divided by x selected as RTC clock, where x:[2,31]
-------------------------	--

Note

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).

Return values

None	
------	--

4.1.5.3.43 RCC_SYSClkConfig()

```
void RCC_SYSClkConfig (
    uint32_t RCC_SYSClkSource )
```

Configures the system clock (SYSClk).

Note

The HSI is used (enabled by hardware) as system clock source after startup from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).

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

Parameters

<i>RCC_SYSClkSource</i>	<p>specifies the clock source used as system clock. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • RCC_SYSClkSource_HSI: HSI selected as system clock source • RCC_SYSClkSource_HSE: HSE selected as system clock source • RCC_SYSClkSource_PLLCLK: PLL selected as system clock source
-------------------------	---

Return values

None	
------	--

4.1.5.3.44 RCC_WaitForHSEStartUp()

```
ErrorStatus RCC_WaitForHSEStartUp (
    void )
```

Waits for HSE start-up.

Note

This functions waits on HSERDY flag to be set and return SUCCESS if this flag is set, otherwise returns ERROR if the timeout is reached and this flag is not set. The timeout value is defined by the constant HSE↔_STARTUP_TIMEOUT in stm32f4xx.h file. You can tailor it depending on the HSE crystal used in your application.

Parameters

None	
------	--

Return values

<i>An</i>	ErrorStatus enumeration value: <ul style="list-style-type: none"> • SUCCESS: HSE oscillator is stable and ready to use • ERROR: HSE oscillator not yet ready
-----------	--

4.1.5.4 RCC_Private_Functions

Modules

- [Internal and external clocks, PLL, CSS and MCO configuration functions](#)
Internal and external clocks, PLL, CSS and MCO configuration functions.
- [System AHB and APB busses clocks configuration functions](#)
System, AHB and APB busses clocks configuration functions.
- [Peripheral clocks configuration functions](#)
Peripheral clocks configuration functions.
- [Interrupts and flags management functions](#)
Interrupts and flags management functions.

4.1.5.4.1 Detailed Description

4.1.5.4.2 Internal and external clocks, PLL, CSS and MCO configuration functions

Internal and external clocks, PLL, CSS and MCO configuration functions.

Functions

- void [RCC_DeInit](#) (void)
Resets the RCC clock configuration to the default reset state.
- void [RCC_HSEConfig](#) (uint8_t RCC_HSE)
Configures the External High Speed oscillator (HSE).
- ErrorStatus [RCC_WaitForHSEStartUp](#) (void)
Waits for HSE start-up.
- void [RCC_AdjustHSICalibrationValue](#) (uint8_t HSICalibrationValue)
Adjusts the Internal High Speed oscillator (HSI) calibration value.
- void [RCC_HSICmd](#) (FunctionalState NewState)
Enables or disables the Internal High Speed oscillator (HSI).
- void [RCC_LSEConfig](#) (uint8_t RCC_LSE)
Configures the External Low Speed oscillator (LSE).
- void [RCC_LSICmd](#) (FunctionalState NewState)
Enables or disables the Internal Low Speed oscillator (LSI).
- void [RCC_PLLConfig](#) (uint32_t RCC_PLLSource, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP, uint32_t PLLQ)
Configures the main PLL clock source, multiplication and division factors.
- void [RCC_PLLCmd](#) (FunctionalState NewState)
Enables or disables the main PLL.
- void [RCC_PLLI2SConfig](#) (uint32_t PLLI2SN, uint32_t PLLI2SR)
Configures the PLLI2S clock multiplication and division factors.
- void [RCC_PLLI2SCmd](#) (FunctionalState NewState)
Enables or disables the PLLI2S.
- void [RCC_ClockSecuritySystemCmd](#) (FunctionalState NewState)
Enables or disables the Clock Security System.

- void `RCC_MCO1Config` (uint32_t RCC_MCO1Source, uint32_t RCC_MCO1Div)
Selects the clock source to output on MCO1 pin(PA8).
- void `RCC_MCO2Config` (uint32_t RCC_MCO2Source, uint32_t RCC_MCO2Div)
Selects the clock source to output on MCO2 pin(PC9).

4.1.5.4.2.1 Detailed Description

Internal and external clocks, PLL, CSS and MCO configuration functions.

```
=====
Internal/external clocks, PLL, CSS and MCO configuration functions
=====
```

This section provide functions allowing to configure the internal/external clocks, PLLs, CSS and MCO pins.

1. HSI (high-speed internal), 16 MHz factory-trimmed RC used directly or through the PLL as System clock source.
2. LSI (low-speed internal), 32 KHz low consumption RC used as IWDG and/or RTC clock source.
3. HSE (high-speed external), 4 to 26 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
4. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
5. PLL (clocked by HSI or HSE), featuring two different output clocks:
 - The first output is used to generate the high speed system clock (up to 168 MHz)
 - The second output is used to generate the clock for the USB OTG FS (48 MHz), the random analog generator (<=48 MHz) and the SDIO (<= 48 MHz).
6. PLLI2S (clocked by HSI or HSE), used to generate an accurate clock to achieve high-quality audio performance on the I2S interface.
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 HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M4 NMI (Non-Maskable Interrupt) exception vector.
8. MCO1 (microcontroller clock output), used to output HSI, LSE, HSE or PLL clock (through a configurable prescaler) on PA8 pin.
9. MCO2 (microcontroller clock output), used to output HSE, PLL, SYSCLK or PLLI2S clock (through a configurable prescaler) on PC9 pin.

4.1.5.4.2.2 Function Documentation

RCC_AdjustHSICalibrationValue()

```
void RCC_AdjustHSICalibrationValue (
    uint8_t HSICalibrationValue )
```

Adjusts the Internal High Speed oscillator (HSI) calibration value.

Note

The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

Parameters

<i>HSICalibrationValue</i>	specifies the calibration trimming value. This parameter must be a number between 0 and 0x1F.
----------------------------	---

Return values

None	
------	--

RCC_ClockSecuritySystemCmd()

```
void RCC_ClockSecuritySystemCmd (
    FunctionalState NewState )
```

Enables or disables the Clock Security System.

Note

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-M4 NMI (Non-Maskable Interrupt) exception vector.

Parameters

<i>NewState</i>	new state of the Clock Security System. This parameter can be: ENABLE or DISABLE.
-----------------	---

Return values

None	
------	--

RCC_DeInit()

```
void RCC_DeInit (
    void )
```

Resets the RCC clock configuration to the default reset state.

Note

The default reset state of the clock configuration is given below:

- HSI ON and used as system clock source
- HSE, PLL and PLLI2S OFF
- AHB, APB1 and APB2 prescaler set to 1.
- CSS, MCO1 and MCO2 OFF
- All interrupts disabled

This function doesn't modify the configuration of the

- Peripheral clocks
- LSI, LSE and RTC clocks

Parameters

None	
------	--

Return values

None	
------	--

RCC_HSEConfig()

```
void RCC_HSEConfig (
    uint8_t RCC_HSE )
```

Configures the External High Speed oscillator (HSE).

Note

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.

Parameters

<i>RCC_HSE</i>	specifies the new state of the HSE. This parameter can be one of the following values: <ul style="list-style-type: none">• 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
----------------	---

Return values

<i>None</i>	
-------------	--

RCC_HSIConfig()

```
void RCC_HSIConfig (
    FunctionalState NewState )
```

Enables or disables the Internal High Speed oscillator (HSI).

Note

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.

Parameters

<i>NewState</i>	new state of the HSI. This parameter can be: ENABLE or DISABLE.
-----------------	---

Note

When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

Return values

<i>None</i>	
-------------	--

RCC_LSEConfig()

```
void RCC_LSEConfig (
    uint8_t RCC_LSE )
```

Configures the External Low Speed oscillator (LSE).

Note

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 `PWR_BackupAccessCmd(ENABLE)` 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.

Parameters

<i>RCC_LSE</i>	specifies the new state of the LSE. This parameter can be one of the following values: <ul style="list-style-type: none">• <code>RCC_LSE_OFF</code>: turn OFF the LSE oscillator, <code>LSERDY</code> flag goes low after 6 LSE oscillator clock cycles.• <code>RCC_LSE_ON</code>: turn ON the LSE oscillator• <code>RCC_LSE_Bypass</code>: LSE oscillator bypassed with external clock
----------------	---

Return values

<i>None</i>	
-------------	--

RCC_LSIcmd()

```
void RCC_LSIcmd (
    FunctionalState NewState )
```

Enables or disables the Internal Low Speed oscillator (LSI).

Note

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.

Parameters

<i>NewState</i>	new state of the LSI. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .
-----------------	--

Note

When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

Return values

<i>None</i>	
-------------	--

RCC_MCO1Config()

```
void RCC_MCO1Config (
    uint32_t RCC_MCO1Source,
    uint32_t RCC_MCO1Div )
```

Selects the clock source to output on MCO1 pin(PA8).

Note

PA8 should be configured in alternate function mode.

Parameters

<i>RCC_MCO1Source</i>	<p>specifies the clock source to output. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_MCO1Source_HSI</code>: HSI clock selected as MCO1 source • <code>RCC_MCO1Source_LSE</code>: LSE clock selected as MCO1 source • <code>RCC_MCO1Source_HSE</code>: HSE clock selected as MCO1 source • <code>RCC_MCO1Source_PLLCLK</code>: main PLL clock selected as MCO1 source
<i>RCC_MCO1Div</i>	<p>specifies the MCO1 prescaler. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_MCO1Div_1</code>: no division applied to MCO1 clock • <code>RCC_MCO1Div_2</code>: division by 2 applied to MCO1 clock • <code>RCC_MCO1Div_3</code>: division by 3 applied to MCO1 clock • <code>RCC_MCO1Div_4</code>: division by 4 applied to MCO1 clock • <code>RCC_MCO1Div_5</code>: division by 5 applied to MCO1 clock

Return values

<i>None</i>	
-------------	--

RCC_MCO2Config()

```
void RCC_MCO2Config (
    uint32_t RCC_MCO2Source,
    uint32_t RCC_MCO2Div )
```

Selects the clock source to output on MCO2 pin(PC9).

Note

PC9 should be configured in alternate function mode.

Parameters

<i>RCC_MCO2Source</i>	specifies the clock source to output. This parameter can be one of the following values: <ul style="list-style-type: none"> • <i>RCC_MCO2Source_SYSCCLK</i>: System clock (SYSCCLK) selected as MCO2 source • <i>RCC_MCO2Source_PLLI2SCLK</i>: PLLI2S clock selected as MCO2 source • <i>RCC_MCO2Source_HSE</i>: HSE clock selected as MCO2 source • <i>RCC_MCO2Source_PLLCLK</i>: main PLL clock selected as MCO2 source
<i>RCC_MCO2Div</i>	specifies the MCO2 prescaler. This parameter can be one of the following values: <ul style="list-style-type: none"> • <i>RCC_MCO2Div_1</i>: no division applied to MCO2 clock • <i>RCC_MCO2Div_2</i>: division by 2 applied to MCO2 clock • <i>RCC_MCO2Div_3</i>: division by 3 applied to MCO2 clock • <i>RCC_MCO2Div_4</i>: division by 4 applied to MCO2 clock • <i>RCC_MCO2Div_5</i>: division by 5 applied to MCO2 clock

Return values

<i>None</i>	
-------------	--

RCC_PLLCmd()

```
void RCC_PLLCmd (
    FunctionalState NewState )
```

Enables or disables the main PLL.

Note

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.

Parameters

<i>NewState</i>	new state of the main PLL. This parameter can be: ENABLE or DISABLE.
-----------------	--

Return values

<i>None</i>	
-------------	--

RCC_PLLConfig()

```
void RCC_PLLConfig (
    uint32_t RCC_PLLSource,
    uint32_t PLLM,
    uint32_t PLLN,
    uint32_t PLLP,
```



```
uint32_t PLLQ )
```

Configures the main PLL clock source, multiplication and division factors.

Note

This function must be used only when the main PLL is disabled.

Parameters

<i>RCC_PLLSource</i>	specifies the PLL entry clock source. This parameter can be one of the following values: <ul style="list-style-type: none"> • <i>RCC_PLLSource_HSI</i>: HSI oscillator clock selected as PLL clock entry • <i>RCC_PLLSource_HSE</i>: HSE oscillator clock selected as PLL clock entry
----------------------	---

Note

This clock source (*RCC_PLLSource*) is common for the main PLL and PLLI2S.

Parameters

<i>PLLM</i>	specifies the division factor for PLL VCO input clock This parameter must be a number between 0 and 63.
-------------	---

Note

You have to set the *PLLM* parameter correctly to ensure that the VCO input frequency ranges from 1 to 2 MHz. It is recommended to select a frequency of 2 MHz to limit PLL jitter.

Parameters

<i>PLLN</i>	specifies the multiplication factor for PLL VCO output clock This parameter must be a number between 192 and 432.
-------------	---

Note

You have to set the *PLLN* parameter correctly to ensure that the VCO output frequency is between 192 and 432 MHz.

Parameters

<i>PLLQ</i>	specifies the division factor for main system clock (SYSCLK) This parameter must be a number in the range {2, 4, 6, or 8}.
-------------	--

Note

You have to set the *PLLQ* parameter correctly to not exceed 168 MHz on the System clock frequency.

Parameters

<i>PLLQ</i>	specifies the division factor for OTG FS, SDIO and RNG clocks This parameter must be a number between 4 and 15.
-------------	---

Note

If the USB OTG FS is used in your application, you have to set the PLLQ parameter correctly to have 48 MHz clock for the USB. However, the SDIO and RNG need a frequency lower than or equal to 48 MHz to work correctly.

Return values

<i>None</i>	
-------------	--

RCC_PLLI2SCmd()

```
void RCC_PLLI2SCmd (
    FunctionalState NewState )
```

Enables or disables the PLLI2S.

Note

The PLLI2S is disabled by hardware when entering STOP and STANDBY modes.

Parameters

<i>NewState</i>	new state of the PLLI2S. This parameter can be: ENABLE or DISABLE.
-----------------	--

Return values

<i>None</i>	
-------------	--

RCC_PLLI2SConfig()

```
void RCC_PLLI2SConfig (
    uint32_t PLLI2SN,
    uint32_t PLLI2SR )
```

Configures the PLLI2S clock multiplication and division factors.

Note

This function must be used only when the PLLI2S is disabled.

PLLI2S clock source is common with the main PLL (configured in RCC_PLLConfig function)

Parameters

<i>PLLI2SN</i>	specifies the multiplication factor for PLLI2S VCO output clock This parameter must be a number between 192 and 432.
----------------	--

Note

You have to set the PLLI2SN parameter correctly to ensure that the VCO output frequency is between 192 and 432 MHz.

Parameters

<i>PLLI2SR</i>	specifies the division factor for I2S clock This parameter must be a number between 2 and 7.
----------------	--

Note

You have to set the PLLI2SR parameter correctly to not exceed 192 MHz on the I2S clock frequency.

Return values

None	
------	--

RCC_WaitForHSEStartUp()

```
ErrorStatus RCC_WaitForHSEStartUp (
    void )
```

Waits for HSE start-up.

Note

This functions waits on HSERDY flag to be set and return SUCCESS if this flag is set, otherwise returns ERROR if the timeout is reached and this flag is not set. The timeout value is defined by the constant HSE↔_STARTUP_TIMEOUT in stm32f4xx.h file. You can tailor it depending on the HSE crystal used in your application.

Parameters

None	
------	--

Return values

An	ErrorStatus enumeration value: <ul style="list-style-type: none"> • SUCCESS: HSE oscillator is stable and ready to use • ERROR: HSE oscillator not yet ready
----	--

4.1.5.4.3 System AHB and APB busses clocks configuration functions

System, AHB and APB busses clocks configuration functions.

Functions

- void [RCC_SYSCLKConfig](#) (uint32_t RCC_SYSCLKSource)
Configures the system clock (SYSCLK).
- uint8_t [RCC_GetSYSCLKSource](#) (void)
Returns the clock source used as system clock.
- void [RCC_HCLKConfig](#) (uint32_t RCC_SYSCLK)
Configures the AHB clock (HCLK).
- void [RCC_PCLK1Config](#) (uint32_t RCC_HCLK)
Configures the Low Speed APB clock (PCLK1).
- void [RCC_PCLK2Config](#) (uint32_t RCC_HCLK)
Configures the High Speed APB clock (PCLK2).
- void [RCC_GetClocksFreq](#) ([RCC_ClocksTypeDef](#) *RCC_Clocks)
Returns the frequencies of different on chip clocks; SYSCLK, HCLK, PCLK1 and PCLK2.

4.1.5.4.3.1 Detailed Description

System, AHB and APB busses clocks configuration functions.

System, AHB and APB busses clocks configuration functions

This section provide functions allowing to configure the System, AHB, APB1 and APB2 busses clocks.

- Several clock sources can be used to drive the System clock (SYSCLK): HSI, HSE and PLL.
The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...).
APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses.
You can use "RCC_GetClocksFreq()" function to retrieve the frequencies of these clocks.

@note All the peripheral clocks are derived from the System clock (SYSCLK) except:

- I2S: the I2S clock can be derived either from a specific PLL (PLL_I2S) or from an external clock mapped on the I2S_CKIN pin.
You have to use RCC_I2SCLKConfig() function to configure this clock.
- RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 31. You have to use RCC_RTCCLKConfig() and RCC_RTCCLKCmd() functions to configure this clock.
- USB OTG FS, SDIO and RTC: USB OTG FS require a frequency equal to 48 MHz to work correctly, while the SDIO require a frequency equal or lower than to 48. This clock is derived of the main PLL through PLLQ divider.
- IWDG clock which is always the LSI clock.

- The maximum frequency of the SYSCLK and HCLK is 168 MHz, PCLK2 82 MHz and PCLK1 42 MHz.
Depending on the device voltage range, the maximum frequency should be adapted accordingly:

Latency	HCLK clock frequency (MHz)			
	voltage range	voltage range	voltage range	voltage range
	2.7 V - 3.6 V	2.4 V - 2.7 V	2.1 V - 2.4 V	1.8 V - 2.1 V
0WS(1CPU cycle)	0 < HCLK <= 30	0 < HCLK <= 24	0 < HCLK <= 18	0 < HCLK <= 16
1WS(2CPU cycle)	30 < HCLK <= 60	24 < HCLK <= 48	18 < HCLK <= 36	16 < HCLK <= 32
2WS(3CPU cycle)	60 < HCLK <= 90	48 < HCLK <= 72	36 < HCLK <= 54	32 < HCLK <= 48
3WS(4CPU cycle)	90 < HCLK <= 120	72 < HCLK <= 96	54 < HCLK <= 72	48 < HCLK <= 64
4WS(5CPU cycle)	120 < HCLK <= 150	96 < HCLK <= 120	72 < HCLK <= 90	64 < HCLK <= 80
5WS(6CPU cycle)	120 < HCLK <= 168	120 < HCLK <= 144	90 < HCLK <= 108	80 < HCLK <= 96
6WS(7CPU cycle)	NA	144 < HCLK <= 168	108 < HCLK <= 120	96 < HCLK <= 112
7WS(8CPU cycle)	NA	NA	120 < HCLK <= 138	112 < HCLK <= 120

@note When VOS bit (in PWR_CR register) is reset to '0, the maximum value of HCLK is 144 MHz.
You can use PWR_MainRegulatorModeConfig() function to set or reset this bit.

4.1.5.4.3.2 Function Documentation

RCC_GetClocksFreq()

```
void RCC_GetClocksFreq (
    RCC_ClocksTypeDef * RCC_Clocks )
```

Returns the frequencies of different on chip clocks; SYSCLK, HCLK, PCLK1 and PCLK2.

Note

The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:

If SYSCLK source is HSI, function returns values based on [HSI_VALUE\(*\)](#)

If SYSCLK source is HSE, function returns values based on [HSE_VALUE\(**\)](#)

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 stm32f4xx.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 stm32f4xx.h file (default value 25 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.

Parameters

<i>RCC_Clocks</i>	pointer to a RCC_ClocksTypeDef structure which will hold the clocks frequencies.
-------------------	--

Note

This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.

Each time SYSCLK, HCLK, PCLK1 and/or PCLK2 clock changes, this function must be called to update the structure's field. Otherwise, any configuration based on this function will be incorrect.

Return values

<i>None</i>	
-------------	--

RCC_GetSYSCLKSource()

```
uint8_t RCC_GetSYSCLKSource (
    void )
```

Returns the clock source used as system clock.

Parameters

<i>None</i>	
-------------	--

Return values

<i>The</i>	clock source used as system clock. The returned value can be one of the following: <ul style="list-style-type: none"> • 0x00: HSI used as system clock • 0x04: HSE used as system clock • 0x08: PLL used as system clock
------------	---

RCC_HCLKConfig()

```
void RCC_HCLKConfig (
    uint32_t RCC_SYSCLK )
```

Configures the AHB clock (HCLK).

Note

Depending on the device voltage range, the software has to set correctly these bits to ensure that HCLK not exceed the maximum allowed frequency (for more details refer to section above "CPU, AHB and APB busses clocks configuration functions")

Parameters

<i>RCC_SYSCLK</i>	<p>defines the AHB clock divider. This clock is derived from the system clock (SYSCLK). This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • RCC_SYSCLK_Div1: AHB clock = SYSCLK • RCC_SYSCLK_Div2: AHB clock = SYSCLK/2 • RCC_SYSCLK_Div4: AHB clock = SYSCLK/4 • RCC_SYSCLK_Div8: AHB clock = SYSCLK/8 • RCC_SYSCLK_Div16: AHB clock = SYSCLK/16 • RCC_SYSCLK_Div64: AHB clock = SYSCLK/64 • RCC_SYSCLK_Div128: AHB clock = SYSCLK/128 • RCC_SYSCLK_Div256: AHB clock = SYSCLK/256 • RCC_SYSCLK_Div512: AHB clock = SYSCLK/512
-------------------	--

Return values

<i>None</i>	
-------------	--

RCC_PCLK1Config()

```
void RCC_PCLK1Config (
    uint32_t RCC_HCLK )
```

Configures the Low Speed APB clock (PCLK1).

Parameters

<i>RCC_HCLK</i>	<p>defines the APB1 clock divider. This clock is derived from the AHB clock (HCLK). This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • RCC_HCLK_Div1: APB1 clock = HCLK • RCC_HCLK_Div2: APB1 clock = HCLK/2 • RCC_HCLK_Div4: APB1 clock = HCLK/4 • RCC_HCLK_Div8: APB1 clock = HCLK/8 • RCC_HCLK_Div16: APB1 clock = HCLK/16
-----------------	---

Return values

None	
------	--

RCC_PCLK2Config()

```
void RCC_PCLK2Config (
    uint32_t RCC_HCLK )
```

Configures the High Speed APB clock (PCLK2).

Parameters

<i>RCC_HCLK</i>	<p>defines the APB2 clock divider. This clock is derived from the AHB clock (HCLK). This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • RCC_HCLK_Div1: APB2 clock = HCLK • RCC_HCLK_Div2: APB2 clock = HCLK/2 • RCC_HCLK_Div4: APB2 clock = HCLK/4 • RCC_HCLK_Div8: APB2 clock = HCLK/8 • RCC_HCLK_Div16: APB2 clock = HCLK/16
-----------------	---

Return values

None	
------	--

RCC_SYSCLKConfig()

```
void RCC_SYSCLKConfig (
    uint32_t RCC_SYSCLKSource )
```

Configures the system clock (SYSCLK).

Note

The HSI is used (enabled by hardware) as system clock source after startup from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).

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

Parameters

<i>RCC_SYSCLKSource</i>	<p>specifies the clock source used as system clock. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • RCC_SYSCLKSource_HSI: HSI selected as system clock source • RCC_SYSCLKSource_HSE: HSE selected as system clock source • RCC_SYSCLKSource_PLLCLK: PLL selected as system clock source
-------------------------	---

Return values

None	
------	--

4.1.5.4.4 Peripheral clocks configuration functions

Peripheral clocks configuration functions.

Functions

- void [RCC_RTCCLKConfig](#) (uint32_t RCC_RTCCLKSource)
Configures the RTC clock (RTCCLK).
- void [RCC_RTCCLKCmd](#) (FunctionalState NewState)
Enables or disables the RTC clock.
- void [RCC_BackupResetCmd](#) (FunctionalState NewState)
Forces or releases the Backup domain reset.
- void [RCC_I2SCLKConfig](#) (uint32_t RCC_I2SCLKSource)
Configures the I2S clock source (I2SCLK).
- void [RCC_AHB1PeriphClockCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
Enables or disables the AHB1 peripheral clock.
- void [RCC_AHB2PeriphClockCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
Enables or disables the AHB2 peripheral clock.
- void [RCC_AHB3PeriphClockCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Enables or disables the AHB3 peripheral clock.
- void [RCC_APB1PeriphClockCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Enables or disables the Low Speed APB (APB1) peripheral clock.
- void [RCC_APB2PeriphClockCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Enables or disables the High Speed APB (APB2) peripheral clock.
- void [RCC_AHB1PeriphResetCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
Forces or releases AHB1 peripheral reset.
- void [RCC_AHB2PeriphResetCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
Forces or releases AHB2 peripheral reset.
- void [RCC_AHB3PeriphResetCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Forces or releases AHB3 peripheral reset.
- void [RCC_APB1PeriphResetCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Forces or releases Low Speed APB (APB1) peripheral reset.
- void [RCC_APB2PeriphResetCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Forces or releases High Speed APB (APB2) peripheral reset.
- void [RCC_AHB1PeriphClockLPModeCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
Enables or disables the AHB1 peripheral clock during Low Power (Sleep) mode.
- void [RCC_AHB2PeriphClockLPModeCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
Enables or disables the AHB2 peripheral clock during Low Power (Sleep) mode.
- void [RCC_AHB3PeriphClockLPModeCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Enables or disables the AHB3 peripheral clock during Low Power (Sleep) mode.
- void [RCC_APB1PeriphClockLPModeCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Enables or disables the APB1 peripheral clock during Low Power (Sleep) mode.
- void [RCC_APB2PeriphClockLPModeCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Enables or disables the APB2 peripheral clock during Low Power (Sleep) mode.

4.1.5.4.4.1 Detailed Description

Peripheral clocks configuration functions.

```
=====
Peripheral clocks configuration functions
=====
```

This section provide functions allowing to configure the Peripheral clocks.

1. The RTC clock which is derived from the LSI, LSE or HSE clock divided by 2 to 31.
2. After restart from Reset or wakeup from STANDBY, all peripherals are off except internal SRAM, Flash and JTAG. Before to start using a peripheral you have to enable its interface clock. You can do this using `RCC_AHBPeriphClockCmd()`, `RCC_APB2PeriphClockCmd()` and `RCC_APB1PeriphClockCmd()` functions.
3. To reset the peripherals configuration (to the default state after device reset) you can use `RCC_AHBPeriphResetCmd()`, `RCC_APB2PeriphResetCmd()` and `RCC_APB1PeriphResetCmd()` functions.
4. To further reduce power consumption in SLEEP mode the peripheral clocks can be disabled prior to executing the WFI or WFE instructions. You can do this using `RCC_AHBPeriphClockLPModeCmd()`, `RCC_APB2PeriphClockLPModeCmd()` and `RCC_APB1PeriphClockLPModeCmd()` functions.

4.1.5.4.4.2 Function Documentation

RCC_AHB1PeriphClockCmd()

```
void RCC_AHB1PeriphClockCmd (
    uint32_t RCC_AHB1Periph,
    FunctionalState NewState )
```

Enables or disables the AHB1 peripheral clock.

Note

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

Parameters

<i>RCC_AHBPeriph</i>	<p>specifies the AHB1 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_AHB1Periph_GPIOA</code>: GPIOA clock • <code>RCC_AHB1Periph_GPIOB</code>: GPIOB clock • <code>RCC_AHB1Periph_GPIOC</code>: GPIOC clock • <code>RCC_AHB1Periph_GPIOD</code>: GPIOD clock • <code>RCC_AHB1Periph_GPIOE</code>: GPIOE clock • <code>RCC_AHB1Periph_GPIOF</code>: GPIOF clock • <code>RCC_AHB1Periph_GPIOG</code>: GPIOG clock • <code>RCC_AHB1Periph_GPIOG</code>: GPIOG clock • <code>RCC_AHB1Periph_GPIOI</code>: GPIOI clock • <code>RCC_AHB1Periph_CRC</code>: CRC clock • <code>RCC_AHB1Periph_BKPSRAM</code>: BKPSRAM interface clock • <code>RCC_AHB1Periph_CCMDATARAMEN</code>: CCM data RAM interface clock • <code>RCC_AHB1Periph_DMA1</code>: DMA1 clock • <code>RCC_AHB1Periph_DMA2</code>: DMA2 clock • <code>RCC_AHB1Periph_ETH_MAC</code>: Ethernet MAC clock • <code>RCC_AHB1Periph_ETH_MAC_Tx</code>: Ethernet Transmission clock • <code>RCC_AHB1Periph_ETH_MAC_Rx</code>: Ethernet Reception clock • <code>RCC_AHB1Periph_ETH_MAC_PTP</code>: Ethernet PTP clock • <code>RCC_AHB1Periph_OTG_HS</code>: USB OTG HS clock • <code>RCC_AHB1Periph_OTG_HS_ULPI</code>: USB OTG HS ULPI clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .

Return values

<i>None</i>	
-------------	--

RCC_AHB1PeriphClockLPModeCmd()

```
void RCC_AHB1PeriphClockLPModeCmd (
    uint32_t RCC_AHB1Periph,
    FunctionalState NewState )
```

Enables or disables the AHB1 peripheral clock during Low Power (Sleep) mode.

Note

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again.

By default, all peripheral clocks are enabled during SLEEP mode.

Parameters

<i>RCC_AHBPeriph</i>	<p>specifies the AHB1 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_AHB1Periph_GPIOA: GPIOA clock • RCC_AHB1Periph_GPIOB: GPIOB clock • RCC_AHB1Periph_GPIOC: GPIOC clock • RCC_AHB1Periph_GPIOD: GPIOD clock • RCC_AHB1Periph_GPIOE: GPIOE clock • RCC_AHB1Periph_GPIOF: GPIOF clock • RCC_AHB1Periph_GPIOG: GPIOG clock • RCC_AHB1Periph_GPIOG: GPIOG clock • RCC_AHB1Periph_GPIOI: GPIOI clock • RCC_AHB1Periph_CRC: CRC clock • RCC_AHB1Periph_BKPSRAM: BKPSRAM interface clock • RCC_AHB1Periph_DMA1: DMA1 clock • RCC_AHB1Periph_DMA2: DMA2 clock • RCC_AHB1Periph_ETH_MAC: Ethernet MAC clock • RCC_AHB1Periph_ETH_MAC_Tx: Ethernet Transmission clock • RCC_AHB1Periph_ETH_MAC_Rx: Ethernet Reception clock • RCC_AHB1Periph_ETH_MAC_PTP: Ethernet PTP clock • RCC_AHB1Periph_OTG_HS: USB OTG HS clock • RCC_AHB1Periph_OTG_HS_ULPI: USB OTG HS ULPI clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_AHB1PeriphResetCmd()

```
void RCC_AHB1PeriphResetCmd (
    uint32_t RCC_AHB1Periph,
    FunctionalState NewState )
```

Forces or releases AHB1 peripheral reset.

Parameters

<i>RCC_AHB1Periph</i>	<p>specifies the AHB1 peripheral to reset. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_AHB1Periph_GPIOA</code>: GPIOA clock • <code>RCC_AHB1Periph_GPIOB</code>: GPIOB clock • <code>RCC_AHB1Periph_GPIOC</code>: GPIOC clock • <code>RCC_AHB1Periph_GPIOD</code>: GPIOD clock • <code>RCC_AHB1Periph_GPIOE</code>: GPIOE clock • <code>RCC_AHB1Periph_GPIOF</code>: GPIOF clock • <code>RCC_AHB1Periph_GPIOG</code>: GPIOG clock • <code>RCC_AHB1Periph_GPIOG</code>: GPIOG clock • <code>RCC_AHB1Periph_GPIOI</code>: GPIOI clock • <code>RCC_AHB1Periph_CRC</code>: CRC clock • <code>RCC_AHB1Periph_DMA1</code>: DMA1 clock • <code>RCC_AHB1Periph_DMA2</code>: DMA2 clock • <code>RCC_AHB1Periph_ETH_MAC</code>: Ethernet MAC clock • <code>RCC_AHB1Periph_OTG_HS</code>: USB OTG HS clock
<i>NewState</i>	new state of the specified peripheral reset. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .

Return values

<i>None</i>	
-------------	--

RCC_AHB2PeriphClockCmd()

```
void RCC_AHB2PeriphClockCmd (
    uint32_t RCC_AHB2Periph,
    FunctionalState NewState )
```

Enables or disables the AHB2 peripheral clock.

Note

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

Parameters

<i>RCC_AHBPeriph</i>	specifies the AHB2 peripheral to gates its clock. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • RCC_AHB2Periph_DCMI: DCMI clock • RCC_AHB2Periph_Cryp: CRYp clock • RCC_AHB2Periph_HASH: HASH clock • RCC_AHB2Periph_RNG: RNG clock • RCC_AHB2Periph_OTG_FS: USB OTG FS clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_AHB2PeriphClockLPModeCmd()

```
void RCC_AHB2PeriphClockLPModeCmd (
    uint32_t RCC_AHB2Periph,
    FunctionalState NewState )
```

Enables or disables the AHB2 peripheral clock during Low Power (Sleep) mode.

Note

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again.

By default, all peripheral clocks are enabled during SLEEP mode.

Parameters

<i>RCC_AHBPeriph</i>	specifies the AHB2 peripheral to gates its clock. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • RCC_AHB2Periph_DCMI: DCMI clock • RCC_AHB2Periph_Cryp: CRYp clock • RCC_AHB2Periph_HASH: HASH clock • RCC_AHB2Periph_RNG: RNG clock • RCC_AHB2Periph_OTG_FS: USB OTG FS clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_AHB2PeriphResetCmd()

```
void RCC_AHB2PeriphResetCmd (
    uint32_t RCC_AHB2Periph,
    FunctionalState NewState )
```

Forces or releases AHB2 peripheral reset.

Parameters

<i>RCC_AHB2Periph</i>	specifies the AHB2 peripheral to reset. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • RCC_AHB2Periph_DCMI: DCMI clock • RCC_AHB2Periph_CRYP: CRYP clock • RCC_AHB2Periph_HASH: HASH clock • RCC_AHB2Periph_RNG: RNG clock • RCC_AHB2Periph_OTG_FS: USB OTG FS clock
<i>NewState</i>	new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_AHB3PeriphClockCmd()

```
void RCC_AHB3PeriphClockCmd (
    uint32_t RCC_AHB3Periph,
    FunctionalState NewState )
```

Enables or disables the AHB3 peripheral clock.

Note

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

Parameters

<i>RCC_AHBPeriph</i>	specifies the AHB3 peripheral to gates its clock. This parameter must be: RCC_AHB3Periph_FSMC
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_AHB3PeriphClockLPModeCmd()

```
void RCC_AHB3PeriphClockLPModeCmd (
    uint32_t RCC_AHB3Periph,
    FunctionalState NewState )
```

Enables or disables the AHB3 peripheral clock during Low Power (Sleep) mode.

Note

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again.

By default, all peripheral clocks are enabled during SLEEP mode.

Parameters

<i>RCC_AHBPeriph</i>	specifies the AHB3 peripheral to gates its clock. This parameter must be: RCC_AHB3Periph_FSMC
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_AHB3PeriphResetCmd()

```
void RCC_AHB3PeriphResetCmd (
    uint32_t RCC_AHB3Periph,
    FunctionalState NewState )
```

Forces or releases AHB3 peripheral reset.

Parameters

<i>RCC_AHB3Periph</i>	specifies the AHB3 peripheral to reset. This parameter must be: RCC_AHB3Periph_FSMC
<i>NewState</i>	new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_APB1PeriphClockCmd()

```
void RCC_APB1PeriphClockCmd (
    uint32_t RCC_APB1Periph,
    FunctionalState NewState )
```

Enables or disables the Low Speed APB (APB1) peripheral clock.

Note

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

Parameters

<i>RCC_APB1Periph</i>	<p>specifies the APB1 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_APB1Periph_TIM2</code>: TIM2 clock • <code>RCC_APB1Periph_TIM3</code>: TIM3 clock • <code>RCC_APB1Periph_TIM4</code>: TIM4 clock • <code>RCC_APB1Periph_TIM5</code>: TIM5 clock • <code>RCC_APB1Periph_TIM6</code>: TIM6 clock • <code>RCC_APB1Periph_TIM7</code>: TIM7 clock • <code>RCC_APB1Periph_TIM12</code>: TIM12 clock • <code>RCC_APB1Periph_TIM13</code>: TIM13 clock • <code>RCC_APB1Periph_TIM14</code>: TIM14 clock • <code>RCC_APB1Periph_WWDG</code>: WWDG clock • <code>RCC_APB1Periph_SPI2</code>: SPI2 clock • <code>RCC_APB1Periph_SPI3</code>: SPI3 clock • <code>RCC_APB1Periph_USART2</code>: USART2 clock • <code>RCC_APB1Periph_USART3</code>: USART3 clock • <code>RCC_APB1Periph_UART4</code>: UART4 clock • <code>RCC_APB1Periph_UART5</code>: UART5 clock • <code>RCC_APB1Periph_I2C1</code>: I2C1 clock • <code>RCC_APB1Periph_I2C2</code>: I2C2 clock • <code>RCC_APB1Periph_I2C3</code>: I2C3 clock • <code>RCC_APB1Periph_CAN1</code>: CAN1 clock • <code>RCC_APB1Periph_CAN2</code>: CAN2 clock • <code>RCC_APB1Periph_PWR</code>: PWR clock • <code>RCC_APB1Periph_DAC</code>: DAC clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .

Return values

<i>None</i>	
-------------	--

RCC_APB1PeriphClockLPModeCmd()

```
void RCC_APB1PeriphClockLPModeCmd (
    uint32_t RCC_APB1Periph,
    FunctionalState NewState )
```

Enables or disables the APB1 peripheral clock during Low Power (Sleep) mode.

Note

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again.

By default, all peripheral clocks are enabled during SLEEP mode.

Parameters

<i>RCC_APB1Periph</i>	<p>specifies the APB1 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_APB1Periph_TIM2: TIM2 clock • RCC_APB1Periph_TIM3: TIM3 clock • RCC_APB1Periph_TIM4: TIM4 clock • RCC_APB1Periph_TIM5: TIM5 clock • RCC_APB1Periph_TIM6: TIM6 clock • RCC_APB1Periph_TIM7: TIM7 clock • RCC_APB1Periph_TIM12: TIM12 clock • RCC_APB1Periph_TIM13: TIM13 clock • RCC_APB1Periph_TIM14: TIM14 clock • RCC_APB1Periph_WWDG: WWDG clock • RCC_APB1Periph_SPI2: SPI2 clock • RCC_APB1Periph_SPI3: SPI3 clock • RCC_APB1Periph_USART2: USART2 clock • RCC_APB1Periph_USART3: USART3 clock • RCC_APB1Periph_UART4: UART4 clock • RCC_APB1Periph_UART5: UART5 clock • RCC_APB1Periph_I2C1: I2C1 clock • RCC_APB1Periph_I2C2: I2C2 clock • RCC_APB1Periph_I2C3: I2C3 clock • RCC_APB1Periph_CAN1: CAN1 clock • RCC_APB1Periph_CAN2: CAN2 clock • RCC_APB1Periph_PWR: PWR clock • RCC_APB1Periph_DAC: DAC clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_APB1PeriphResetCmd()

```
void RCC_APB1PeriphResetCmd (
    uint32_t RCC_APB1Periph,
    FunctionalState NewState )
```

Forces or releases Low Speed APB (APB1) peripheral reset.

Parameters

<i>RCC_APB1Periph</i>	<p>specifies the APB1 peripheral to reset. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_APB1Periph_TIM2: TIM2 clock • RCC_APB1Periph_TIM3: TIM3 clock • RCC_APB1Periph_TIM4: TIM4 clock • RCC_APB1Periph_TIM5: TIM5 clock • RCC_APB1Periph_TIM6: TIM6 clock • RCC_APB1Periph_TIM7: TIM7 clock • RCC_APB1Periph_TIM12: TIM12 clock • RCC_APB1Periph_TIM13: TIM13 clock • RCC_APB1Periph_TIM14: TIM14 clock • RCC_APB1Periph_WWDG: WWDG clock • RCC_APB1Periph_SPI2: SPI2 clock • RCC_APB1Periph_SPI3: SPI3 clock • RCC_APB1Periph_USART2: USART2 clock • RCC_APB1Periph_USART3: USART3 clock • RCC_APB1Periph_UART4: UART4 clock • RCC_APB1Periph_UART5: UART5 clock • RCC_APB1Periph_I2C1: I2C1 clock • RCC_APB1Periph_I2C2: I2C2 clock • RCC_APB1Periph_I2C3: I2C3 clock • RCC_APB1Periph_CAN1: CAN1 clock • RCC_APB1Periph_CAN2: CAN2 clock • RCC_APB1Periph_PWR: PWR clock • RCC_APB1Periph_DAC: DAC clock
<i>NewState</i>	new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_APB2PeriphClockCmd()

```
void RCC_APB2PeriphClockCmd (
    uint32_t RCC_APB2Periph,
    FunctionalState NewState )
```

Enables or disables the High Speed APB (APB2) peripheral clock.

Note

After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

Parameters

<i>RCC_APB2Periph</i>	<p>specifies the APB2 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_APB2Periph_TIM1: TIM1 clock • RCC_APB2Periph_TIM8: TIM8 clock • RCC_APB2Periph_USART1: USART1 clock • RCC_APB2Periph_USART6: USART6 clock • RCC_APB2Periph_ADC1: ADC1 clock • RCC_APB2Periph_ADC2: ADC2 clock • RCC_APB2Periph_ADC3: ADC3 clock • RCC_APB2Periph_SDIO: SDIO clock • RCC_APB2Periph_SPI1: SPI1 clock • RCC_APB2Periph_SYSCFG: SYSCFG clock • RCC_APB2Periph_TIM9: TIM9 clock • RCC_APB2Periph_TIM10: TIM10 clock • RCC_APB2Periph_TIM11: TIM11 clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_APB2PeriphClockLPModeCmd()

```
void RCC_APB2PeriphClockLPModeCmd (
    uint32_t RCC_APB2Periph,
    FunctionalState NewState )
```

Enables or disables the APB2 peripheral clock during Low Power (Sleep) mode.

Note

Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again.

By default, all peripheral clocks are enabled during SLEEP mode.

Parameters

<i>RCC_APB2Periph</i>	<p>specifies the APB2 peripheral to gates its clock. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_APB2Periph_TIM1</code>: TIM1 clock • <code>RCC_APB2Periph_TIM8</code>: TIM8 clock • <code>RCC_APB2Periph_USART1</code>: USART1 clock • <code>RCC_APB2Periph_USART6</code>: USART6 clock • <code>RCC_APB2Periph_ADC1</code>: ADC1 clock • <code>RCC_APB2Periph_ADC2</code>: ADC2 clock • <code>RCC_APB2Periph_ADC3</code>: ADC3 clock • <code>RCC_APB2Periph_SDIO</code>: SDIO clock • <code>RCC_APB2Periph_SPI1</code>: SPI1 clock • <code>RCC_APB2Periph_SYSCFG</code>: SYSCFG clock • <code>RCC_APB2Periph_TIM9</code>: TIM9 clock • <code>RCC_APB2Periph_TIM10</code>: TIM10 clock • <code>RCC_APB2Periph_TIM11</code>: TIM11 clock
<i>NewState</i>	new state of the specified peripheral clock. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .

Return values

<i>None</i>	
-------------	--

RCC_APB2PeriphResetCmd()

```
void RCC_APB2PeriphResetCmd (
    uint32_t RCC_APB2Periph,
    FunctionalState NewState )
```

Forces or releases High Speed APB (APB2) peripheral reset.

Parameters

<i>RCC_APB2Periph</i>	<p>specifies the APB2 peripheral to reset. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_APB2Periph_TIM1: TIM1 clock • RCC_APB2Periph_TIM8: TIM8 clock • RCC_APB2Periph_USART1: USART1 clock • RCC_APB2Periph_USART6: USART6 clock • RCC_APB2Periph_ADC1: ADC1 clock • RCC_APB2Periph_ADC2: ADC2 clock • RCC_APB2Periph_ADC3: ADC3 clock • RCC_APB2Periph_SDIO: SDIO clock • RCC_APB2Periph_SPI1: SPI1 clock • RCC_APB2Periph_SYSCFG: SYSCFG clock • RCC_APB2Periph_TIM9: TIM9 clock • RCC_APB2Periph_TIM10: TIM10 clock • RCC_APB2Periph_TIM11: TIM11 clock
<i>NewState</i>	new state of the specified peripheral reset. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

RCC_BackupResetCmd()

```
void RCC_BackupResetCmd (
    FunctionalState NewState )
```

Forces or releases the Backup domain reset.

Note

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.

Parameters

<i>NewState</i>	new state of the Backup domain reset. This parameter can be: ENABLE or DISABLE.
-----------------	---

Return values

<i>None</i>	
-------------	--

RCC_I2SCLKConfig()

```
void RCC_I2SCLKConfig (
    uint32_t RCC_I2SCLKSource )
```

Configures the I2S clock source (I2SCLK).

Note

This function must be called before enabling the I2S APB clock.

Parameters

<i>RCC_I2SCLKSource</i>	<p>specifies the I2S clock source. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_I2S2CLKSource_PLLI2S</code>: PLLI2S clock used as I2S clock source • <code>RCC_I2S2CLKSource_Ext</code>: External clock mapped on the I2S_CKIN pin used as I2S clock source
-------------------------	--

Return values

<i>None</i>	
-------------	--

RCC_RTCCLKCmd()

```
void RCC_RTCCLKCmd (
    FunctionalState NewState )
```

Enables or disables the RTC clock.

Note

This function must be used only after the RTC clock source was selected using the `RCC_RTCCLKConfig` function.

Parameters

<i>NewState</i>	new state of the RTC clock. This parameter can be: <code>ENABLE</code> or <code>DISABLE</code> .
-----------------	--

Return values

<i>None</i>	
-------------	--

RCC_RTCCLKConfig()

```
void RCC_RTCCLKConfig (
    uint32_t RCC_RTCCLKSource )
```

Configures the RTC clock (RTCCLK).

Note

As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using `PWR_BackupAccessCmd(ENABLE)` function before to configure the RTC clock source (to be done once after reset).

Once the RTC clock is configured it can't be changed unless the Backup domain is reset using [RCC_BackupResetCmd\(\)](#) function, or by a Power On Reset (POR).

Parameters

<i>RCC_RTCCLKSource</i>	specifies the RTC clock source. This parameter can be one of the following values: <ul style="list-style-type: none"> • <code>RCC_RTCCLKSource_LSE</code>: LSE selected as RTC clock • <code>RCC_RTCCLKSource_LSI</code>: LSI selected as RTC clock • <code>RCC_RTCCLKSource_HSE_Divx</code>: HSE clock divided by x selected as RTC clock, where x:[2,31]
-------------------------	---

Note

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).

Return values

<i>None</i>	
-------------	--

4.1.5.4.5 Interrupts and flags management functions

Interrupts and flags management functions.

Functions

- void [RCC_ITConfig](#) (uint8_t RCC_IT, FunctionalState NewState)
Enables or disables the specified RCC interrupts.
- FlagStatus [RCC_GetFlagStatus](#) (uint8_t RCC_FLAG)
Checks whether the specified RCC flag is set or not.
- void [RCC_ClearFlag](#) (void)
Clears the RCC reset flags. The reset flags are: `RCC_FLAG_PINRST`, `RCC_FLAG_PORRST`, `RCC_FLAG_SFTRST`, `RCC_FLAG_IWDGRST`, `RCC_FLAG_WWDGRST`, `RCC_FLAG_LPWRRST`.
- ITStatus [RCC_GetITStatus](#) (uint8_t RCC_IT)
Checks whether the specified RCC interrupt has occurred or not.
- void [RCC_ClearITPendingBit](#) (uint8_t RCC_IT)
Clears the RCC's interrupt pending bits.

4.1.5.4.5.1 Detailed Description

Interrupts and flags management functions.

```
=====
                        Interrupts and flags management functions
=====
```

4.1.5.4.5.2 Function Documentation

RCC_ClearFlag()

```
void RCC_ClearFlag (
    void )
```

Clears the RCC reset flags. The reset flags are: `RCC_FLAG_PINRST`, `RCC_FLAG_PORRST`, `RCC_FLAG_SFTRST`, `RCC_FLAG_IWDGRST`, `RCC_FLAG_WWDGRST`, `RCC_FLAG_LPWRRST`.

Parameters

<i>None</i>	
-------------	--

Return values

<i>None</i>	
-------------	--

RCC_ClearITPendingBit()

```
void RCC_ClearITPendingBit (
    uint8_t RCC_IT )
```

Clears the RCC's interrupt pending bits.

Parameters

<i>RCC_IT</i>	<p>specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_IT_LSIRDY: LSI ready interrupt • RCC_IT_LSERDY: LSE ready interrupt • RCC_IT_HSIIRDY: HSI ready interrupt • RCC_IT_HSERDY: HSE ready interrupt • RCC_IT_PLLRDY: main PLL ready interrupt • RCC_IT_PLLI2SRDY: PLLI2S ready interrupt • RCC_IT_CSS: Clock Security System interrupt
---------------	---

Return values

<i>None</i>	
-------------	--

RCC_GetFlagStatus()

```
FlagStatus RCC_GetFlagStatus (
    uint8_t RCC_FLAG )
```

Checks whether the specified RCC flag is set or not.

Parameters

<i>RCC_FLAG</i>	<p>specifies the flag to check. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_FLAG_HSIRDY</code>: HSI oscillator clock ready • <code>RCC_FLAG_HSERDY</code>: HSE oscillator clock ready • <code>RCC_FLAG_PLLRDY</code>: main PLL clock ready • <code>RCC_FLAG_PLLI2SRDY</code>: PLLI2S clock ready • <code>RCC_FLAG_LSERDY</code>: LSE oscillator clock ready • <code>RCC_FLAG_LSIRDY</code>: LSI oscillator clock ready • <code>RCC_FLAG BORRST</code>: POR/PDR or BOR reset • <code>RCC_FLAG_PINRST</code>: Pin reset • <code>RCC_FLAG_PORRST</code>: POR/PDR reset • <code>RCC_FLAG_SFTRST</code>: Software reset • <code>RCC_FLAG_IWDGRST</code>: Independent Watchdog reset • <code>RCC_FLAG_WWDGRST</code>: Window Watchdog reset • <code>RCC_FLAG_LPWRST</code>: Low Power reset
-----------------	--

Return values

<i>The</i>	new state of <code>RCC_FLAG</code> (SET or RESET).
------------	--

RCC_GetITStatus()

```
ITStatus RCC_GetITStatus (
    uint8_t RCC_IT )
```

Checks whether the specified RCC interrupt has occurred or not.

Parameters

<i>RCC_IT</i>	<p>specifies the RCC interrupt source to check. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • <code>RCC_IT_LSIRDY</code>: LSI ready interrupt • <code>RCC_IT_LSERDY</code>: LSE ready interrupt • <code>RCC_IT_HSIRDY</code>: HSI ready interrupt • <code>RCC_IT_HSERDY</code>: HSE ready interrupt • <code>RCC_IT_PLLRDY</code>: main PLL ready interrupt • <code>RCC_IT_PLLI2SRDY</code>: PLLI2S ready interrupt • <code>RCC_IT_CSS</code>: Clock Security System interrupt
---------------	--

Return values

<i>The</i>	new state of RCC_IT (SET or RESET).
------------	-------------------------------------

RCC_ITConfig()

```
void RCC_ITConfig (
    uint8_t RCC_IT,
    FunctionalState NewState )
```

Enables or disables the specified RCC interrupts.

Parameters

<i>RCC_IT</i>	<p>specifies the RCC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • RCC_IT_LSIRDY: LSI ready interrupt • RCC_IT_LSERDY: LSE ready interrupt • RCC_IT_HSIRDY: HSI ready interrupt • RCC_IT_HSERDY: HSE ready interrupt • RCC_IT_PLLRDY: main PLL ready interrupt • RCC_IT_PLLI2SRDY: PLLI2S ready interrupt
<i>NewState</i>	new state of the specified RCC interrupts. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.5.5 RCC_Exported_Constants**Modules**

- [RCC_HSE_configuration](#)
- [RCC_PLL_Clock_Source](#)
- [RCC_System_Clock_Source](#)
- [RCC_AHB_Clock_Source](#)
- [RCC_APB1_APB2_Clock_Source](#)
- [RCC_Interrupt_Source](#)
- [RCC_LSE_Configuration](#)
- [RCC_RTC_Clock_Source](#)
- [RCC_I2S_Clock_Source](#)
- [RCC_AHB1_Peripherals](#)
- [RCC_AHB2_Peripherals](#)
- [RCC_AHB3_Peripherals](#)
- [RCC_APB1_Peripherals](#)
- [RCC_APB2_Peripherals](#)
- [RCC_MCO1_Clock_Source_Prescaler](#)
- [RCC_MCO2_Clock_Source_Prescaler](#)
- [RCC_Flag](#)

4.1.5.5.1 Detailed Description

4.1.5.5.2 RCC_HSE_configuration

Macros

- #define [RCC_HSE_OFF](#) ((uint8_t)0x00)
- #define [RCC_HSE_ON](#) ((uint8_t)0x01)
- #define [RCC_HSE_Bypass](#) ((uint8_t)0x05)
- #define [IS_RCC_HSE](#)(HSE)

4.1.5.5.2.1 Detailed Description

4.1.5.5.2.2 Macro Definition Documentation

IS_RCC_HSE

```
#define IS_RCC_HSE(  
    HSE )
```

Value:

```
((HSE) == RCC\_HSE\_OFF) || ((HSE) == RCC\_HSE\_ON) || \  
((HSE) == RCC\_HSE\_Bypass)
```

RCC_HSE_Bypass

```
#define RCC_HSE_Bypass ((uint8_t)0x05)
```

RCC_HSE_OFF

```
#define RCC_HSE_OFF ((uint8_t)0x00)
```

RCC_HSE_ON

```
#define RCC_HSE_ON ((uint8_t)0x01)
```

4.1.5.5.3 RCC_PLL_Clock_Source

Macros

- #define [RCC_PLLSource_HSI](#) ((uint32_t)0x00000000)
- #define [RCC_PLLSource_HSE](#) ((uint32_t)0x00400000)
- #define [IS_RCC_PLL_SOURCE](#)(SOURCE)
- #define [IS_RCC_PLLM_VALUE](#)(VALUE) ((VALUE) <= 63)
- #define [IS_RCC_PLLN_VALUE](#)(VALUE) ((192 <= (VALUE)) && ((VALUE) <= 432))
- #define [IS_RCC_PLLP_VALUE](#)(VALUE) (((VALUE) == 2) || ((VALUE) == 4) || ((VALUE) == 6) || ((VALUE) == 8))
- #define [IS_RCC_PLLQ_VALUE](#)(VALUE) ((4 <= (VALUE)) && ((VALUE) <= 15))
- #define [IS_RCC_PLLI2SN_VALUE](#)(VALUE) ((192 <= (VALUE)) && ((VALUE) <= 432))
- #define [IS_RCC_PLLI2SR_VALUE](#)(VALUE) ((2 <= (VALUE)) && ((VALUE) <= 7))

4.1.5.5.3.1 Detailed Description

4.1.5.5.3.2 Macro Definition Documentation

IS_RCC_PLL_SOURCE

```
#define IS_RCC_PLL_SOURCE(  
    SOURCE )
```

Value:

```
((SOURCE) == RCC\_PLLSource\_HSI) || \  
((SOURCE) == RCC\_PLLSource\_HSE)
```

IS_RCC_PLLI2SN_VALUE

```
#define IS_RCC_PLLI2SN_VALUE(  
    VALUE ) ((192 <= (VALUE)) && ((VALUE) <= 432))
```

IS_RCC_PLLI2SR_VALUE

```
#define IS_RCC_PLLI2SR_VALUE(  
    VALUE ) ((2 <= (VALUE)) && ((VALUE) <= 7))
```

IS_RCC_PLLM_VALUE

```
#define IS_RCC_PLLM_VALUE(  
    VALUE ) ((VALUE) <= 63)
```

IS_RCC_PLLN_VALUE

```
#define IS_RCC_PLLN_VALUE(  
    VALUE ) ((192 <= (VALUE)) && ((VALUE) <= 432))
```

IS_RCC_PLLP_VALUE

```
#define IS_RCC_PLLP_VALUE(  
    VALUE ) ((VALUE) == 2) || ((VALUE) == 4) || ((VALUE) == 6) || ((VALUE) == 8))
```

IS_RCC_PLLQ_VALUE

```
#define IS_RCC_PLLQ_VALUE(  
    VALUE ) ((4 <= (VALUE)) && ((VALUE) <= 15))
```

RCC_PLLSource_HSE

```
#define RCC_PLLSource_HSE ((uint32_t)0x00400000)
```

RCC_PLLSource_HSI

```
#define RCC_PLLSource_HSI ((uint32_t)0x00000000)
```

4.1.5.5.4 RCC_System_Clock_Source**Macros**

- #define [RCC_SYSCLKSource_HSI](#) ((uint32_t)0x00000000)
- #define [RCC_SYSCLKSource_HSE](#) ((uint32_t)0x00000001)
- #define [RCC_SYSCLKSource_PLLCLK](#) ((uint32_t)0x00000002)
- #define [IS_RCC_SYSCLK_SOURCE](#)(SOURCE)

4.1.5.5.4.1 Detailed Description**4.1.5.5.4.2 Macro Definition Documentation****IS_RCC_SYSCLK_SOURCE**

```
#define IS_RCC_SYSCLK_SOURCE(  
    SOURCE )
```

Value:

```
((SOURCE) == RCC\_SYSCLKSource\_HSI) || \  
((SOURCE) == RCC\_SYSCLKSource\_HSE) || \  
((SOURCE) == RCC\_SYSCLKSource\_PLLCLK))
```

RCC_SYSCLKSource_HSE

```
#define RCC_SYSCLKSource_HSE ((uint32_t)0x00000001)
```

RCC_SYSClkSource_HSI

```
#define RCC_SYSClkSource_HSI ((uint32_t)0x00000000)
```

RCC_SYSClkSource_PLLCLK

```
#define RCC_SYSClkSource_PLLCLK ((uint32_t)0x00000002)
```

4.1.5.5.5 RCC_AHB_Clock_Source**Macros**

- #define **RCC_SYSClk_Div1** ((uint32_t)0x00000000)
- #define **RCC_SYSClk_Div2** ((uint32_t)0x00000080)
- #define **RCC_SYSClk_Div4** ((uint32_t)0x00000090)
- #define **RCC_SYSClk_Div8** ((uint32_t)0x000000A0)
- #define **RCC_SYSClk_Div16** ((uint32_t)0x000000B0)
- #define **RCC_SYSClk_Div64** ((uint32_t)0x000000C0)
- #define **RCC_SYSClk_Div128** ((uint32_t)0x000000D0)
- #define **RCC_SYSClk_Div256** ((uint32_t)0x000000E0)
- #define **RCC_SYSClk_Div512** ((uint32_t)0x000000F0)
- #define **IS_RCC_HCLK**(HCLK)

4.1.5.5.5.1 Detailed Description**4.1.5.5.5.2 Macro Definition Documentation****IS_RCC_HCLK**

```
#define IS_RCC_HCLK(  
    HCLK )
```

Value:

```
((HCLK) == RCC_SYSClk_Div1) || ((HCLK) == RCC_SYSClk_Div2) || \
((HCLK) == RCC_SYSClk_Div4) || ((HCLK) == RCC_SYSClk_Div8) || \
((HCLK) == RCC_SYSClk_Div16) || ((HCLK) == RCC_SYSClk_Div64) || \
((HCLK) == RCC_SYSClk_Div128) || ((HCLK) == RCC_SYSClk_Div256) || \
((HCLK) == RCC_SYSClk_Div512)
```

RCC_SYSClk_Div1

```
#define RCC_SYSClk_Div1 ((uint32_t)0x00000000)
```

RCC_SYSClk_Div128

```
#define RCC_SYSClk_Div128 ((uint32_t)0x000000D0)
```

RCC_SYSClk_Div16

```
#define RCC_SYSClk_Div16 ((uint32_t)0x000000B0)
```

RCC_SYSClk_Div2

```
#define RCC_SYSClk_Div2 ((uint32_t)0x00000080)
```

RCC_SYSClk_Div256

```
#define RCC_SYSClk_Div256 ((uint32_t)0x000000E0)
```

RCC_SYSClk_Div4

```
#define RCC_SYSClk_Div4 ((uint32_t)0x00000090)
```

RCC_SYSCLK_Div512

```
#define RCC_SYSCLK_Div512 ((uint32_t)0x000000F0)
```

RCC_SYSCLK_Div64

```
#define RCC_SYSCLK_Div64 ((uint32_t)0x000000C0)
```

RCC_SYSCLK_Div8

```
#define RCC_SYSCLK_Div8 ((uint32_t)0x000000A0)
```

4.1.5.5.6 RCC_APB1_APB2_Clock_Source**Macros**

- #define [RCC_HCLK_Div1](#) ((uint32_t)0x00000000)
- #define [RCC_HCLK_Div2](#) ((uint32_t)0x00001000)
- #define [RCC_HCLK_Div4](#) ((uint32_t)0x00001400)
- #define [RCC_HCLK_Div8](#) ((uint32_t)0x00001800)
- #define [RCC_HCLK_Div16](#) ((uint32_t)0x00001C00)
- #define [IS_RCC_PCLK](#)(PCLK)

4.1.5.5.6.1 Detailed Description**4.1.5.5.6.2 Macro Definition Documentation****IS_RCC_PCLK**

```
#define IS_RCC_PCLK(  
    PCLK )
```

Value:

```
((PCLK) == RCC\_HCLK\_Div1) || ((PCLK) == RCC\_HCLK\_Div2) || \  
((PCLK) == RCC\_HCLK\_Div4) || ((PCLK) == RCC\_HCLK\_Div8) || \  
((PCLK) == RCC\_HCLK\_Div16)
```

RCC_HCLK_Div1

```
#define RCC_HCLK_Div1 ((uint32_t)0x00000000)
```

RCC_HCLK_Div16

```
#define RCC_HCLK_Div16 ((uint32_t)0x00001C00)
```

RCC_HCLK_Div2

```
#define RCC_HCLK_Div2 ((uint32_t)0x00001000)
```

RCC_HCLK_Div4

```
#define RCC_HCLK_Div4 ((uint32_t)0x00001400)
```

RCC_HCLK_Div8

```
#define RCC_HCLK_Div8 ((uint32_t)0x00001800)
```

4.1.5.5.7 RCC_Interrupt_Source

Macros

- #define `RCC_IT_LSIRDY` ((uint8_t)0x01)
- #define `RCC_IT_LSERDY` ((uint8_t)0x02)
- #define `RCC_IT_HSIRDY` ((uint8_t)0x04)
- #define `RCC_IT_HSERDY` ((uint8_t)0x08)
- #define `RCC_IT_PLLRDY` ((uint8_t)0x10)
- #define `RCC_IT_PLLI2SRDY` ((uint8_t)0x20)
- #define `RCC_IT_CSS` ((uint8_t)0x80)
- #define `IS_RCC_IT(IT)` (((IT) & (uint8_t)0xC0) == 0x00) && ((IT) != 0x00)
- #define `IS_RCC_GET_IT(IT)`
- #define `IS_RCC_CLEAR_IT(IT)` (((IT) & (uint8_t)0x40) == 0x00) && ((IT) != 0x00)

4.1.5.5.7.1 Detailed Description

4.1.5.5.7.2 Macro Definition Documentation

IS_RCC_CLEAR_IT

```
#define IS_RCC_CLEAR_IT(  
    IT ) (((IT) & (uint8_t)0x40) == 0x00) && ((IT) != 0x00)
```

IS_RCC_GET_IT

```
#define IS_RCC_GET_IT(  
    IT )
```

Value:

```
((IT) == RCC_IT_LSIRDY) || ((IT) == RCC_IT_LSERDY) || \  
((IT) == RCC_IT_HSIRDY) || ((IT) == RCC_IT_HSERDY) || \  
((IT) == RCC_IT_PLLRDY) || ((IT) == RCC_IT_CSS) || \  
((IT) == RCC_IT_PLLI2SRDY)
```

IS_RCC_IT

```
#define IS_RCC_IT(  
    IT ) (((IT) & (uint8_t)0xC0) == 0x00) && ((IT) != 0x00)
```

RCC_IT_CSS

```
#define RCC_IT_CSS ((uint8_t)0x80)
```

RCC_IT_HSERDY

```
#define RCC_IT_HSERDY ((uint8_t)0x08)
```

RCC_IT_HSIRDY

```
#define RCC_IT_HSIRDY ((uint8_t)0x04)
```

RCC_IT_LSERDY

```
#define RCC_IT_LSERDY ((uint8_t)0x02)
```

RCC_IT_LSIRDY

```
#define RCC_IT_LSIRDY ((uint8_t)0x01)
```

RCC_IT_PLLI2SRDY

```
#define RCC_IT_PLLI2SRDY ((uint8_t)0x20)
```

RCC_IT_PLLRDY

```
#define RCC_IT_PLLRDY ((uint8_t)0x10)
```

4.1.5.5.8 RCC_LSE_Configuration**Macros**

- #define [RCC_LSE_OFF](#) ((uint8_t)0x00)
- #define [RCC_LSE_ON](#) ((uint8_t)0x01)
- #define [RCC_LSE_Bypass](#) ((uint8_t)0x04)
- #define [IS_RCC_LSE](#)(LSE)

4.1.5.5.8.1 Detailed Description**4.1.5.5.8.2 Macro Definition Documentation****IS_RCC_LSE**

```
#define IS_RCC_LSE(  
    LSE )
```

Value:

```
((LSE) == RCC\_LSE\_OFF) || ((LSE) == RCC\_LSE\_ON) || \  
((LSE) == RCC\_LSE\_Bypass)
```

RCC_LSE_Bypass

```
#define RCC_LSE_Bypass ((uint8_t)0x04)
```

RCC_LSE_OFF

```
#define RCC_LSE_OFF ((uint8_t)0x00)
```

RCC_LSE_ON

```
#define RCC_LSE_ON ((uint8_t)0x01)
```

4.1.5.5.9 RCC_RTC_Clock_Source**Macros**

- #define [RCC_RTCCLKSource_LSE](#) ((uint32_t)0x00000100)
- #define [RCC_RTCCLKSource_LSI](#) ((uint32_t)0x00000200)
- #define [RCC_RTCCLKSource_HSE_Div2](#) ((uint32_t)0x00020300)
- #define [RCC_RTCCLKSource_HSE_Div3](#) ((uint32_t)0x00030300)
- #define [RCC_RTCCLKSource_HSE_Div4](#) ((uint32_t)0x00040300)
- #define [RCC_RTCCLKSource_HSE_Div5](#) ((uint32_t)0x00050300)
- #define [RCC_RTCCLKSource_HSE_Div6](#) ((uint32_t)0x00060300)
- #define [RCC_RTCCLKSource_HSE_Div7](#) ((uint32_t)0x00070300)
- #define [RCC_RTCCLKSource_HSE_Div8](#) ((uint32_t)0x00080300)
- #define [RCC_RTCCLKSource_HSE_Div9](#) ((uint32_t)0x00090300)
- #define [RCC_RTCCLKSource_HSE_Div10](#) ((uint32_t)0x000A0300)
- #define [RCC_RTCCLKSource_HSE_Div11](#) ((uint32_t)0x000B0300)
- #define [RCC_RTCCLKSource_HSE_Div12](#) ((uint32_t)0x000C0300)
- #define [RCC_RTCCLKSource_HSE_Div13](#) ((uint32_t)0x000D0300)
- #define [RCC_RTCCLKSource_HSE_Div14](#) ((uint32_t)0x000E0300)
- #define [RCC_RTCCLKSource_HSE_Div15](#) ((uint32_t)0x000F0300)
- #define [RCC_RTCCLKSource_HSE_Div16](#) ((uint32_t)0x00100300)
- #define [RCC_RTCCLKSource_HSE_Div17](#) ((uint32_t)0x00110300)
- #define [RCC_RTCCLKSource_HSE_Div18](#) ((uint32_t)0x00120300)

- `#define RCC_RTCCLKSource_HSE_Div19 ((uint32_t)0x00130300)`
- `#define RCC_RTCCLKSource_HSE_Div20 ((uint32_t)0x00140300)`
- `#define RCC_RTCCLKSource_HSE_Div21 ((uint32_t)0x00150300)`
- `#define RCC_RTCCLKSource_HSE_Div22 ((uint32_t)0x00160300)`
- `#define RCC_RTCCLKSource_HSE_Div23 ((uint32_t)0x00170300)`
- `#define RCC_RTCCLKSource_HSE_Div24 ((uint32_t)0x00180300)`
- `#define RCC_RTCCLKSource_HSE_Div25 ((uint32_t)0x00190300)`
- `#define RCC_RTCCLKSource_HSE_Div26 ((uint32_t)0x001A0300)`
- `#define RCC_RTCCLKSource_HSE_Div27 ((uint32_t)0x001B0300)`
- `#define RCC_RTCCLKSource_HSE_Div28 ((uint32_t)0x001C0300)`
- `#define RCC_RTCCLKSource_HSE_Div29 ((uint32_t)0x001D0300)`
- `#define RCC_RTCCLKSource_HSE_Div30 ((uint32_t)0x001E0300)`
- `#define RCC_RTCCLKSource_HSE_Div31 ((uint32_t)0x001F0300)`
- `#define IS_RCC_RTCCLK_SOURCE(SOURCE)`

4.1.5.5.9.1 Detailed Description

4.1.5.5.9.2 Macro Definition Documentation

IS_RCC_RTCCLK_SOURCE

```
#define IS_RCC_RTCCLK_SOURCE(  
    SOURCE )
```

RCC_RTCCLKSource_HSE_Div10

```
#define RCC_RTCCLKSource_HSE_Div10 ((uint32_t)0x000A0300)
```

RCC_RTCCLKSource_HSE_Div11

```
#define RCC_RTCCLKSource_HSE_Div11 ((uint32_t)0x000B0300)
```

RCC_RTCCLKSource_HSE_Div12

```
#define RCC_RTCCLKSource_HSE_Div12 ((uint32_t)0x000C0300)
```

RCC_RTCCLKSource_HSE_Div13

```
#define RCC_RTCCLKSource_HSE_Div13 ((uint32_t)0x000D0300)
```

RCC_RTCCLKSource_HSE_Div14

```
#define RCC_RTCCLKSource_HSE_Div14 ((uint32_t)0x000E0300)
```

RCC_RTCCLKSource_HSE_Div15

```
#define RCC_RTCCLKSource_HSE_Div15 ((uint32_t)0x000F0300)
```

RCC_RTCCLKSource_HSE_Div16

```
#define RCC_RTCCLKSource_HSE_Div16 ((uint32_t)0x00100300)
```

RCC_RTCCLKSource_HSE_Div17

```
#define RCC_RTCCLKSource_HSE_Div17 ((uint32_t)0x00110300)
```

RCC_RTCCLKSource_HSE_Div18

```
#define RCC_RTCCLKSource_HSE_Div18 ((uint32_t)0x00120300)
```

RCC_RTCCLKSource_HSE_Div19

```
#define RCC_RTCCLKSource_HSE_Div19 ((uint32_t)0x00130300)
```

RCC_RTCCLKSource_HSE_Div2

```
#define RCC_RTCCLKSource_HSE_Div2 ((uint32_t)0x00020300)
```

RCC_RTCCLKSource_HSE_Div20

```
#define RCC_RTCCLKSource_HSE_Div20 ((uint32_t)0x00140300)
```

RCC_RTCCLKSource_HSE_Div21

```
#define RCC_RTCCLKSource_HSE_Div21 ((uint32_t)0x00150300)
```

RCC_RTCCLKSource_HSE_Div22

```
#define RCC_RTCCLKSource_HSE_Div22 ((uint32_t)0x00160300)
```

RCC_RTCCLKSource_HSE_Div23

```
#define RCC_RTCCLKSource_HSE_Div23 ((uint32_t)0x00170300)
```

RCC_RTCCLKSource_HSE_Div24

```
#define RCC_RTCCLKSource_HSE_Div24 ((uint32_t)0x00180300)
```

RCC_RTCCLKSource_HSE_Div25

```
#define RCC_RTCCLKSource_HSE_Div25 ((uint32_t)0x00190300)
```

RCC_RTCCLKSource_HSE_Div26

```
#define RCC_RTCCLKSource_HSE_Div26 ((uint32_t)0x001A0300)
```

RCC_RTCCLKSource_HSE_Div27

```
#define RCC_RTCCLKSource_HSE_Div27 ((uint32_t)0x001B0300)
```

RCC_RTCCLKSource_HSE_Div28

```
#define RCC_RTCCLKSource_HSE_Div28 ((uint32_t)0x001C0300)
```

RCC_RTCCLKSource_HSE_Div29

```
#define RCC_RTCCLKSource_HSE_Div29 ((uint32_t)0x001D0300)
```

RCC_RTCCLKSource_HSE_Div3

```
#define RCC_RTCCLKSource_HSE_Div3 ((uint32_t)0x00030300)
```

RCC_RTCCLKSource_HSE_Div30

```
#define RCC_RTCCLKSource_HSE_Div30 ((uint32_t)0x001E0300)
```

RCC_RTCCLKSource_HSE_Div31

```
#define RCC_RTCCLKSource_HSE_Div31 ((uint32_t)0x001F0300)
```

RCC_RTCCLKSource_HSE_Div4

```
#define RCC_RTCCLKSource_HSE_Div4 ((uint32_t)0x00040300)
```

RCC_RTCCLKSource_HSE_Div5

```
#define RCC_RTCCLKSource_HSE_Div5 ((uint32_t)0x00050300)
```

RCC_RTCCLKSource_HSE_Div6

```
#define RCC_RTCCLKSource_HSE_Div6 ((uint32_t)0x00060300)
```

RCC_RTCCLKSource_HSE_Div7

```
#define RCC_RTCCLKSource_HSE_Div7 ((uint32_t)0x00070300)
```

RCC_RTCCLKSource_HSE_Div8

```
#define RCC_RTCCLKSource_HSE_Div8 ((uint32_t)0x00080300)
```

RCC_RTCCLKSource_HSE_Div9

```
#define RCC_RTCCLKSource_HSE_Div9 ((uint32_t)0x00090300)
```

RCC_RTCCLKSource_LSE

```
#define RCC_RTCCLKSource_LSE ((uint32_t)0x00000100)
```

RCC_RTCCLKSource_LSI

```
#define RCC_RTCCLKSource_LSI ((uint32_t)0x00000200)
```

4.1.5.5.10 RCC_I2S_Clock_Source**Macros**

- #define [RCC_I2S2CLKSource_PLLI2S](#) ((uint8_t)0x00)
- #define [RCC_I2S2CLKSource_Ext](#) ((uint8_t)0x01)
- #define [IS_RCC_I2SCLK_SOURCE](#)(SOURCE) (((SOURCE) == [RCC_I2S2CLKSource_PLLI2S](#)) || ((SOURCE) == [RCC_I2S2CLKSource_Ext](#)))

4.1.5.5.10.1 Detailed Description**4.1.5.5.10.2 Macro Definition Documentation****IS_RCC_I2SCLK_SOURCE**

```
#define IS_RCC_I2SCLK_SOURCE(  
    SOURCE ) (((SOURCE) == RCC\_I2S2CLKSource\_PLLI2S) || ((SOURCE) == RCC\_I2S2CLKSource\_Ext))
```

RCC_I2S2CLKSource_Ext

```
#define RCC_I2S2CLKSource_Ext ((uint8_t)0x01)
```

RCC_I2S2CLKSource_PLLI2S

```
#define RCC_I2S2CLKSource_PLLI2S ((uint8_t)0x00)
```

4.1.5.5.11 RCC_AHB1_Peripherals

Macros

- #define [RCC_AHB1Periph_GPIOA](#) ((uint32_t)0x00000001)
- #define [RCC_AHB1Periph_GPIOB](#) ((uint32_t)0x00000002)
- #define [RCC_AHB1Periph_GPIOC](#) ((uint32_t)0x00000004)
- #define [RCC_AHB1Periph_GPIOD](#) ((uint32_t)0x00000008)
- #define [RCC_AHB1Periph_GPIOE](#) ((uint32_t)0x00000010)
- #define [RCC_AHB1Periph_GPIOF](#) ((uint32_t)0x00000020)
- #define [RCC_AHB1Periph_GPIOG](#) ((uint32_t)0x00000040)
- #define [RCC_AHB1Periph_GPIOH](#) ((uint32_t)0x00000080)
- #define [RCC_AHB1Periph_GPIOI](#) ((uint32_t)0x00000100)
- #define [RCC_AHB1Periph_CRC](#) ((uint32_t)0x00001000)
- #define [RCC_AHB1Periph_FLITF](#) ((uint32_t)0x00008000)
- #define [RCC_AHB1Periph_SRAM1](#) ((uint32_t)0x00010000)
- #define [RCC_AHB1Periph_SRAM2](#) ((uint32_t)0x00020000)
- #define [RCC_AHB1Periph_BKPSRAM](#) ((uint32_t)0x00040000)
- #define [RCC_AHB1Periph_CCMDATARAMEN](#) ((uint32_t)0x00100000)
- #define [RCC_AHB1Periph_DMA1](#) ((uint32_t)0x00200000)
- #define [RCC_AHB1Periph_DMA2](#) ((uint32_t)0x00400000)
- #define [RCC_AHB1Periph_ETH_MAC](#) ((uint32_t)0x02000000)
- #define [RCC_AHB1Periph_ETH_MAC_Tx](#) ((uint32_t)0x04000000)
- #define [RCC_AHB1Periph_ETH_MAC_Rx](#) ((uint32_t)0x08000000)
- #define [RCC_AHB1Periph_ETH_MAC_PTP](#) ((uint32_t)0x10000000)
- #define [RCC_AHB1Periph_OTG_HS](#) ((uint32_t)0x20000000)
- #define [RCC_AHB1Periph_OTG_HS_ULPI](#) ((uint32_t)0x40000000)
- #define [IS_RCC_AHB1_CLOCK_PERIPH](#)(PERIPH) (((PERIPH) & 0x818BEE00) == 0x00) && ((PERIPH) != 0x00)
- #define [IS_RCC_AHB1_RESET_PERIPH](#)(PERIPH) (((PERIPH) & 0xDD9FEE00) == 0x00) && ((PERIPH) != 0x00)
- #define [IS_RCC_AHB1_LPMODE_PERIPH](#)(PERIPH) (((PERIPH) & 0x81986E00) == 0x00) && ((PERIPH) != 0x00)

4.1.5.5.11.1 Detailed Description

4.1.5.5.11.2 Macro Definition Documentation

IS_RCC_AHB1_CLOCK_PERIPH

```
#define IS_RCC_AHB1_CLOCK_PERIPH(  
    PERIPH ) (((PERIPH) & 0x818BEE00) == 0x00) && ((PERIPH) != 0x00)
```

IS_RCC_AHB1_LPMODE_PERIPH

```
#define IS_RCC_AHB1_LPMODE_PERIPH(  
    PERIPH ) (((PERIPH) & 0x81986E00) == 0x00) && ((PERIPH) != 0x00)
```

IS_RCC_AHB1_RESET_PERIPH

```
#define IS_RCC_AHB1_RESET_PERIPH(  
    PERIPH ) (((PERIPH) & 0xDD9FEE00) == 0x00) && ((PERIPH) != 0x00)
```

RCC_AHB1Periph_BKPSRAM

```
#define RCC_AHB1Periph_BKPSRAM ((uint32_t)0x00040000)
```

RCC_AHB1Periph_CCMDATARAMEN

```
#define RCC_AHB1Periph_CCMDATARAMEN ((uint32_t)0x00100000)
```

RCC_AHB1Periph_CRC

```
#define RCC_AHB1Periph_CRC ((uint32_t)0x00001000)
```

RCC_AHB1Periph_DMA1

```
#define RCC_AHB1Periph_DMA1 ((uint32_t)0x00200000)
```

RCC_AHB1Periph_DMA2

```
#define RCC_AHB1Periph_DMA2 ((uint32_t)0x00400000)
```

RCC_AHB1Periph_ETH_MAC

```
#define RCC_AHB1Periph_ETH_MAC ((uint32_t)0x02000000)
```

RCC_AHB1Periph_ETH_MAC_PTP

```
#define RCC_AHB1Periph_ETH_MAC_PTP ((uint32_t)0x10000000)
```

RCC_AHB1Periph_ETH_MAC_Rx

```
#define RCC_AHB1Periph_ETH_MAC_Rx ((uint32_t)0x08000000)
```

RCC_AHB1Periph_ETH_MAC_Tx

```
#define RCC_AHB1Periph_ETH_MAC_Tx ((uint32_t)0x04000000)
```

RCC_AHB1Periph_FLITF

```
#define RCC_AHB1Periph_FLITF ((uint32_t)0x00008000)
```

RCC_AHB1Periph_GPIOA

```
#define RCC_AHB1Periph_GPIOA ((uint32_t)0x00000001)
```

RCC_AHB1Periph_GPIOB

```
#define RCC_AHB1Periph_GPIOB ((uint32_t)0x00000002)
```

RCC_AHB1Periph_GPIOC

```
#define RCC_AHB1Periph_GPIOC ((uint32_t)0x00000004)
```

RCC_AHB1Periph_GPIOD

```
#define RCC_AHB1Periph_GPIOD ((uint32_t)0x00000008)
```

RCC_AHB1Periph_GPIOE

```
#define RCC_AHB1Periph_GPIOE ((uint32_t)0x00000010)
```

RCC_AHB1Periph_GPIOF

```
#define RCC_AHB1Periph_GPIOF ((uint32_t)0x00000020)
```

RCC_AHB1Periph_GPIOG

```
#define RCC_AHB1Periph_GPIOG ((uint32_t)0x00000040)
```

RCC_AHB1Periph_GPIOH

```
#define RCC_AHB1Periph_GPIOH ((uint32_t)0x00000080)
```

RCC_AHB1Periph_GPIOI

```
#define RCC_AHB1Periph_GPIOI ((uint32_t)0x00000100)
```

RCC_AHB1Periph_OTG_HS

```
#define RCC_AHB1Periph_OTG_HS ((uint32_t)0x20000000)
```

RCC_AHB1Periph_OTG_HS_ULPI

```
#define RCC_AHB1Periph_OTG_HS_ULPI ((uint32_t)0x40000000)
```

RCC_AHB1Periph_SRAM1

```
#define RCC_AHB1Periph_SRAM1 ((uint32_t)0x00010000)
```

RCC_AHB1Periph_SRAM2

```
#define RCC_AHB1Periph_SRAM2 ((uint32_t)0x00020000)
```

4.1.5.5.12 RCC_AHB2_Peripherals**Macros**

- [#define RCC_AHB2Periph_DCMI](#) ((uint32_t)0x00000001)
- [#define RCC_AHB2Periph_Cryp](#) ((uint32_t)0x00000010)
- [#define RCC_AHB2Periph_HASH](#) ((uint32_t)0x00000020)
- [#define RCC_AHB2Periph_RNG](#) ((uint32_t)0x00000040)
- [#define RCC_AHB2Periph_OTG_FS](#) ((uint32_t)0x00000080)
- [#define IS_RCC_AHB2_PERIPH](#)(PERIPH) (((PERIPH) & 0xFFFFF0E) == 0x00) && ((PERIPH) != 0x00)

4.1.5.5.12.1 Detailed Description**4.1.5.5.12.2 Macro Definition Documentation****IS_RCC_AHB2_PERIPH**

```
#define IS_RCC_AHB2_PERIPH(  
    PERIPH ) (((PERIPH) & 0xFFFFF0E) == 0x00) && ((PERIPH) != 0x00)
```

RCC_AHB2Periph_Cryp

```
#define RCC_AHB2Periph_Cryp ((uint32_t)0x00000010)
```

RCC_AHB2Periph_DCMI

```
#define RCC_AHB2Periph_DCMI ((uint32_t)0x00000001)
```

RCC_AHB2Periph_HASH

```
#define RCC_AHB2Periph_HASH ((uint32_t)0x00000020)
```

RCC_AHB2Periph_OTG_FS

```
#define RCC_AHB2Periph_OTG_FS ((uint32_t)0x00000080)
```

RCC_AHB2Periph_RNG

```
#define RCC_AHB2Periph_RNG ((uint32_t)0x00000040)
```

4.1.5.5.13 RCC_AHB3_Peripherals**Macros**

- #define [RCC_AHB3Periph_FSMC](#) ((uint32_t)0x00000001)
- #define [IS_RCC_AHB3_PERIPH](#)(PERIPH) (((PERIPH) & 0xFFFFFFF0) == 0x00) && ((PERIPH) != 0x00)

4.1.5.5.13.1 Detailed Description**4.1.5.5.13.2 Macro Definition Documentation****IS_RCC_AHB3_PERIPH**

```
#define IS_RCC_AHB3_PERIPH(  
    PERIPH ) (((PERIPH) & 0xFFFFFFF0) == 0x00) && ((PERIPH) != 0x00)
```

RCC_AHB3Periph_FSMC

```
#define RCC_AHB3Periph_FSMC ((uint32_t)0x00000001)
```

4.1.5.5.14 RCC_APB1_Peripherals**Macros**

- #define [RCC_APB1Periph_TIM2](#) ((uint32_t)0x00000001)
- #define [RCC_APB1Periph_TIM3](#) ((uint32_t)0x00000002)
- #define [RCC_APB1Periph_TIM4](#) ((uint32_t)0x00000004)
- #define [RCC_APB1Periph_TIM5](#) ((uint32_t)0x00000008)
- #define [RCC_APB1Periph_TIM6](#) ((uint32_t)0x00000010)
- #define [RCC_APB1Periph_TIM7](#) ((uint32_t)0x00000020)
- #define [RCC_APB1Periph_TIM12](#) ((uint32_t)0x00000040)
- #define [RCC_APB1Periph_TIM13](#) ((uint32_t)0x00000080)
- #define [RCC_APB1Periph_TIM14](#) ((uint32_t)0x00000100)
- #define [RCC_APB1Periph_WWDG](#) ((uint32_t)0x00000800)
- #define [RCC_APB1Periph_SPI2](#) ((uint32_t)0x00004000)
- #define [RCC_APB1Periph_SPI3](#) ((uint32_t)0x00008000)
- #define [RCC_APB1Periph_USART2](#) ((uint32_t)0x00020000)
- #define [RCC_APB1Periph_USART3](#) ((uint32_t)0x00040000)
- #define [RCC_APB1Periph_UART4](#) ((uint32_t)0x00080000)
- #define [RCC_APB1Periph_UART5](#) ((uint32_t)0x00100000)
- #define [RCC_APB1Periph_I2C1](#) ((uint32_t)0x00200000)
- #define [RCC_APB1Periph_I2C2](#) ((uint32_t)0x00400000)
- #define [RCC_APB1Periph_I2C3](#) ((uint32_t)0x00800000)
- #define [RCC_APB1Periph_CAN1](#) ((uint32_t)0x02000000)
- #define [RCC_APB1Periph_CAN2](#) ((uint32_t)0x04000000)
- #define [RCC_APB1Periph_PWR](#) ((uint32_t)0x10000000)
- #define [RCC_APB1Periph_DAC](#) ((uint32_t)0x20000000)
- #define [IS_RCC_APB1_PERIPH](#)(PERIPH) (((PERIPH) & 0xC9013600) == 0x00) && ((PERIPH) != 0x00)

4.1.5.5.14.1 Detailed Description**4.1.5.5.14.2 Macro Definition Documentation****IS_RCC_APB1_PERIPH**

```
#define IS_RCC_APB1_PERIPH(  
    PERIPH ) (((PERIPH) & 0xC9013600) == 0x00) && ((PERIPH) != 0x00)
```

RCC_APB1Periph_CAN1

```
#define RCC_APB1Periph_CAN1 ((uint32_t)0x02000000)
```

RCC_APB1Periph_CAN2

```
#define RCC_APB1Periph_CAN2 ((uint32_t)0x04000000)
```

RCC_APB1Periph_DAC

```
#define RCC_APB1Periph_DAC ((uint32_t)0x20000000)
```

RCC_APB1Periph_I2C1

```
#define RCC_APB1Periph_I2C1 ((uint32_t)0x00200000)
```

RCC_APB1Periph_I2C2

```
#define RCC_APB1Periph_I2C2 ((uint32_t)0x00400000)
```

RCC_APB1Periph_I2C3

```
#define RCC_APB1Periph_I2C3 ((uint32_t)0x00800000)
```

RCC_APB1Periph_PWR

```
#define RCC_APB1Periph_PWR ((uint32_t)0x10000000)
```

RCC_APB1Periph_SPI2

```
#define RCC_APB1Periph_SPI2 ((uint32_t)0x00004000)
```

RCC_APB1Periph_SPI3

```
#define RCC_APB1Periph_SPI3 ((uint32_t)0x00008000)
```

RCC_APB1Periph_TIM12

```
#define RCC_APB1Periph_TIM12 ((uint32_t)0x00000040)
```

RCC_APB1Periph_TIM13

```
#define RCC_APB1Periph_TIM13 ((uint32_t)0x00000080)
```

RCC_APB1Periph_TIM14

```
#define RCC_APB1Periph_TIM14 ((uint32_t)0x00000100)
```

RCC_APB1Periph_TIM2

```
#define RCC_APB1Periph_TIM2 ((uint32_t)0x00000001)
```

RCC_APB1Periph_TIM3

```
#define RCC_APB1Periph_TIM3 ((uint32_t)0x00000002)
```

RCC_APB1Periph_TIM4

```
#define RCC_APB1Periph_TIM4 ((uint32_t)0x00000004)
```


RCC_APB1Periph_TIM5

```
#define RCC_APB1Periph_TIM5 ((uint32_t)0x00000008)
```

RCC_APB1Periph_TIM6

```
#define RCC_APB1Periph_TIM6 ((uint32_t)0x00000010)
```

RCC_APB1Periph_TIM7

```
#define RCC_APB1Periph_TIM7 ((uint32_t)0x00000020)
```

RCC_APB1Periph_UART4

```
#define RCC_APB1Periph_UART4 ((uint32_t)0x00080000)
```

RCC_APB1Periph_UART5

```
#define RCC_APB1Periph_UART5 ((uint32_t)0x00100000)
```

RCC_APB1Periph_USART2

```
#define RCC_APB1Periph_USART2 ((uint32_t)0x00020000)
```

RCC_APB1Periph_USART3

```
#define RCC_APB1Periph_USART3 ((uint32_t)0x00040000)
```

RCC_APB1Periph_WWDG

```
#define RCC_APB1Periph_WWDG ((uint32_t)0x00000800)
```

4.1.5.5.15 RCC_APB2_Peripherals**Macros**

- #define [RCC_APB2Periph_TIM1](#) ((uint32_t)0x00000001)
- #define [RCC_APB2Periph_TIM8](#) ((uint32_t)0x00000002)
- #define [RCC_APB2Periph_USART1](#) ((uint32_t)0x00000010)
- #define [RCC_APB2Periph_USART6](#) ((uint32_t)0x00000020)
- #define [RCC_APB2Periph_ADC](#) ((uint32_t)0x00000100)
- #define [RCC_APB2Periph_ADC1](#) ((uint32_t)0x00000100)
- #define [RCC_APB2Periph_ADC2](#) ((uint32_t)0x00000200)
- #define [RCC_APB2Periph_ADC3](#) ((uint32_t)0x00000400)
- #define [RCC_APB2Periph_SDIO](#) ((uint32_t)0x00000800)
- #define [RCC_APB2Periph_SPI1](#) ((uint32_t)0x00001000)
- #define [RCC_APB2Periph_SYSCFG](#) ((uint32_t)0x00004000)
- #define [RCC_APB2Periph_TIM9](#) ((uint32_t)0x00010000)
- #define [RCC_APB2Periph_TIM10](#) ((uint32_t)0x00020000)
- #define [RCC_APB2Periph_TIM11](#) ((uint32_t)0x00040000)
- #define [IS_RCC_APB2_PERIPH](#)(PERIPH) (((PERIPH) & 0xFFFF8A0CC) == 0x00) && ((PERIPH) != 0x00)
- #define [IS_RCC_APB2_RESET_PERIPH](#)(PERIPH) (((PERIPH) & 0xFFFF8A6CC) == 0x00) && ((PERIPH) != 0x00)

4.1.5.5.15.1 Detailed Description**4.1.5.5.15.2 Macro Definition Documentation****IS_RCC_APB2_PERIPH**

```
#define IS_RCC_APB2_PERIPH(  
    PERIPH ) (((PERIPH) & 0xFFFF8A0CC) == 0x00) && ((PERIPH) != 0x00)
```

IS_RCC_APB2_RESET_PERIPH

```
#define IS_RCC_APB2_RESET_PERIPH(  
    PERIPH ) (((PERIPH) & 0xFFF8A6CC) == 0x00) && ((PERIPH) != 0x00)
```

RCC_APB2Periph_ADC

```
#define RCC_APB2Periph_ADC ((uint32_t)0x00000100)
```

RCC_APB2Periph_ADC1

```
#define RCC_APB2Periph_ADC1 ((uint32_t)0x00000100)
```

RCC_APB2Periph_ADC2

```
#define RCC_APB2Periph_ADC2 ((uint32_t)0x00000200)
```

RCC_APB2Periph_ADC3

```
#define RCC_APB2Periph_ADC3 ((uint32_t)0x00000400)
```

RCC_APB2Periph_SDIO

```
#define RCC_APB2Periph_SDIO ((uint32_t)0x00000800)
```

RCC_APB2Periph_SPI1

```
#define RCC_APB2Periph_SPI1 ((uint32_t)0x00001000)
```

RCC_APB2Periph_SYSCFG

```
#define RCC_APB2Periph_SYSCFG ((uint32_t)0x00004000)
```

RCC_APB2Periph_TIM1

```
#define RCC_APB2Periph_TIM1 ((uint32_t)0x00000001)
```

RCC_APB2Periph_TIM10

```
#define RCC_APB2Periph_TIM10 ((uint32_t)0x00020000)
```

RCC_APB2Periph_TIM11

```
#define RCC_APB2Periph_TIM11 ((uint32_t)0x00040000)
```

RCC_APB2Periph_TIM8

```
#define RCC_APB2Periph_TIM8 ((uint32_t)0x00000002)
```

RCC_APB2Periph_TIM9

```
#define RCC_APB2Periph_TIM9 ((uint32_t)0x00010000)
```

RCC_APB2Periph_USART1

```
#define RCC_APB2Periph_USART1 ((uint32_t)0x00000010)
```

RCC_APB2Periph_USART6

```
#define RCC_APB2Periph_USART6 ((uint32_t)0x00000020)
```

4.1.5.5.16 RCC_MCO1_Clock_Source_Prescaler

Macros

- #define [RCC_MCO1Source_HSI](#) ((uint32_t)0x00000000)
- #define [RCC_MCO1Source_LSE](#) ((uint32_t)0x00200000)
- #define [RCC_MCO1Source_HSE](#) ((uint32_t)0x00400000)
- #define [RCC_MCO1Source_PLLCLK](#) ((uint32_t)0x00600000)
- #define [RCC_MCO1Div_1](#) ((uint32_t)0x00000000)
- #define [RCC_MCO1Div_2](#) ((uint32_t)0x04000000)
- #define [RCC_MCO1Div_3](#) ((uint32_t)0x05000000)
- #define [RCC_MCO1Div_4](#) ((uint32_t)0x06000000)
- #define [RCC_MCO1Div_5](#) ((uint32_t)0x07000000)
- #define [IS_RCC_MCO1SOURCE](#)(SOURCE)
- #define [IS_RCC_MCO1DIV](#)(DIV)

4.1.5.5.16.1 Detailed Description

4.1.5.5.16.2 Macro Definition Documentation

IS_RCC_MCO1DIV

```
#define IS_RCC_MCO1DIV(  
    DIV )
```

Value:

```
((DIV) == RCC\_MCO1Div\_1) || ((DIV) == RCC\_MCO1Div\_2) || \  
((DIV) == RCC\_MCO1Div\_3) || ((DIV) == RCC\_MCO1Div\_4) || \  
((DIV) == RCC\_MCO1Div\_5)
```

IS_RCC_MCO1SOURCE

```
#define IS_RCC_MCO1SOURCE(  
    SOURCE )
```

Value:

```
((SOURCE) == RCC\_MCO1Source\_HSI) || ((SOURCE) == RCC\_MCO1Source\_LSE) || \  
((SOURCE) == RCC\_MCO1Source\_HSE) || ((SOURCE) == RCC\_MCO1Source\_PLLCLK)
```

RCC_MCO1Div_1

```
#define RCC_MCO1Div_1 ((uint32_t)0x00000000)
```

RCC_MCO1Div_2

```
#define RCC_MCO1Div_2 ((uint32_t)0x04000000)
```

RCC_MCO1Div_3

```
#define RCC_MCO1Div_3 ((uint32_t)0x05000000)
```

RCC_MCO1Div_4

```
#define RCC_MCO1Div_4 ((uint32_t)0x06000000)
```

RCC_MCO1Div_5

```
#define RCC_MCO1Div_5 ((uint32_t)0x07000000)
```

RCC_MCO1Source_HSE

```
#define RCC_MCO1Source_HSE ((uint32_t)0x00400000)
```

RCC_MCO1Source_HSI

```
#define RCC_MCO1Source_HSI ((uint32_t)0x00000000)
```

RCC_MCO1Source_LSE

```
#define RCC_MCO1Source_LSE ((uint32_t)0x00200000)
```

RCC_MCO1Source_PLLCLK

```
#define RCC_MCO1Source_PLLCLK ((uint32_t)0x00600000)
```

4.1.5.5.17 RCC_MCO2_Clock_Source_Prescaler**Macros**

- `#define RCC_MCO2Source_SYSCLK ((uint32_t)0x00000000)`
- `#define RCC_MCO2Source_PLLI2SCLK ((uint32_t)0x40000000)`
- `#define RCC_MCO2Source_HSE ((uint32_t)0x80000000)`
- `#define RCC_MCO2Source_PLLCLK ((uint32_t)0xC0000000)`
- `#define RCC_MCO2Div_1 ((uint32_t)0x00000000)`
- `#define RCC_MCO2Div_2 ((uint32_t)0x20000000)`
- `#define RCC_MCO2Div_3 ((uint32_t)0x28000000)`
- `#define RCC_MCO2Div_4 ((uint32_t)0x30000000)`
- `#define RCC_MCO2Div_5 ((uint32_t)0x38000000)`
- `#define IS_RCC_MCO2SOURCE(SOURCE)`
- `#define IS_RCC_MCO2DIV(DIV)`

4.1.5.5.17.1 Detailed Description**4.1.5.5.17.2 Macro Definition Documentation****IS_RCC_MCO2DIV**

```
#define IS_RCC_MCO2DIV(  
    DIV )
```

Value:

```
((DIV) == RCC_MCO2Div_1) || ((DIV) == RCC_MCO2Div_2) || \
((DIV) == RCC_MCO2Div_3) || ((DIV) == RCC_MCO2Div_4) || \
((DIV) == RCC_MCO2Div_5)
```

IS_RCC_MCO2SOURCE

```
#define IS_RCC_MCO2SOURCE(  
    SOURCE )
```

Value:

```
((SOURCE) == RCC_MCO2Source_SYSCLK) || ((SOURCE) ==  
RCC_MCO2Source_PLLI2SCLK) || \
((SOURCE) == RCC_MCO2Source_HSE) || ((SOURCE) == RCC_MCO2Source_PLLCLK)
```

RCC_MCO2Div_1

```
#define RCC_MCO2Div_1 ((uint32_t)0x00000000)
```

RCC_MCO2Div_2

```
#define RCC_MCO2Div_2 ((uint32_t)0x20000000)
```

RCC_MCO2Div_3

```
#define RCC_MCO2Div_3 ((uint32_t)0x28000000)
```

RCC_MCO2Div_4

```
#define RCC_MCO2Div_4 ((uint32_t)0x30000000)
```

RCC_MCO2Div_5

```
#define RCC_MCO2Div_5 ((uint32_t)0x38000000)
```

RCC_MCO2Source_HSE

```
#define RCC_MCO2Source_HSE ((uint32_t)0x80000000)
```

RCC_MCO2Source_PLLCLK

```
#define RCC_MCO2Source_PLLCLK ((uint32_t)0xC0000000)
```

RCC_MCO2Source_PLLI2SCLK

```
#define RCC_MCO2Source_PLLI2SCLK ((uint32_t)0x40000000)
```

RCC_MCO2Source_SYSCLK

```
#define RCC_MCO2Source_SYSCLK ((uint32_t)0x00000000)
```

4.1.5.5.18 RCC_Flag**Macros**

- #define [RCC_FLAG_HSIRDY](#) ((uint8_t)0x21)
- #define [RCC_FLAG_HSERDY](#) ((uint8_t)0x31)
- #define [RCC_FLAG_PLLRDY](#) ((uint8_t)0x39)
- #define [RCC_FLAG_PLLI2SRDY](#) ((uint8_t)0x3B)
- #define [RCC_FLAG_LSERDY](#) ((uint8_t)0x41)
- #define [RCC_FLAG_LSIRDY](#) ((uint8_t)0x61)
- #define [RCC_FLAG BORRST](#) ((uint8_t)0x79)
- #define [RCC_FLAG_PINRST](#) ((uint8_t)0x7A)
- #define [RCC_FLAG_PORRST](#) ((uint8_t)0x7B)
- #define [RCC_FLAG_SFTRST](#) ((uint8_t)0x7C)
- #define [RCC_FLAG_IWDGRST](#) ((uint8_t)0x7D)
- #define [RCC_FLAG_WWDGRST](#) ((uint8_t)0x7E)
- #define [RCC_FLAG_LPWRRST](#) ((uint8_t)0x7F)
- #define [IS_RCC_FLAG](#)(FLAG)
- #define [IS_RCC_CALIBRATION_VALUE](#)(VALUE) ((VALUE) <= 0x1F)

4.1.5.5.18.1 Detailed Description**4.1.5.5.18.2 Macro Definition Documentation****IS_RCC_CALIBRATION_VALUE**

```
#define IS_RCC_CALIBRATION_VALUE(  
    VALUE ) ((VALUE) <= 0x1F)
```

IS_RCC_FLAG

```
#define IS_RCC_FLAG(  
    FLAG )
```

Value:

```
((FLAG) == RCC_FLAG_HSIRDY) || ((FLAG) == RCC_FLAG_HSERDY) || \
((FLAG) == RCC_FLAG_PLLRDY) || ((FLAG) == RCC_FLAG_LSERDY) || \
((FLAG) == RCC_FLAG_LSIRDY) || ((FLAG) == RCC_FLAG_BORRST) || \
((FLAG) == RCC_FLAG_PINRST) || ((FLAG) == RCC_FLAG_PORRST) || \
((FLAG) == RCC_FLAG_SFTRST) || ((FLAG) == RCC_FLAG_IWDGRST) || \
((FLAG) == RCC_FLAG_WWDGRST) || ((FLAG) == RCC_FLAG_LPWRST) || \
((FLAG) == RCC_FLAG_PLLI2SRDY)
```

RCC_FLAG_BORRST

```
#define RCC_FLAG_BORRST ((uint8_t)0x79)
```

RCC_FLAG_HSERDY

```
#define RCC_FLAG_HSERDY ((uint8_t)0x31)
```

RCC_FLAG_HSIRDY

```
#define RCC_FLAG_HSIRDY ((uint8_t)0x21)
```

RCC_FLAG_IWDGRST

```
#define RCC_FLAG_IWDGRST ((uint8_t)0x7D)
```

RCC_FLAG_LPWRST

```
#define RCC_FLAG_LPWRST ((uint8_t)0x7F)
```

RCC_FLAG_LSERDY

```
#define RCC_FLAG_LSERDY ((uint8_t)0x41)
```

RCC_FLAG_LSIRDY

```
#define RCC_FLAG_LSIRDY ((uint8_t)0x61)
```

RCC_FLAG_PINRST

```
#define RCC_FLAG_PINRST ((uint8_t)0x7A)
```

RCC_FLAG_PLLI2SRDY

```
#define RCC_FLAG_PLLI2SRDY ((uint8_t)0x3B)
```

RCC_FLAG_PLLRDY

```
#define RCC_FLAG_PLLRDY ((uint8_t)0x39)
```

RCC_FLAG_PORRST

```
#define RCC_FLAG_PORRST ((uint8_t)0x7B)
```

RCC_FLAG_SFTRST

```
#define RCC_FLAG_SFTRST ((uint8_t)0x7C)
```

RCC_FLAG_WWDGRST

```
#define RCC_FLAG_WWDGRST ((uint8_t)0x7E)
```

4.1.6 USART

USART driver modules.

Modules

- [USART_Private_Functions](#)
- [USART_Exported_Constants](#)

Data Structures

- struct [USART_InitTypeDef](#)
USART Init Structure definition
- struct [USART_ClockInitTypeDef](#)
USART Clock Init Structure definition

Macros

- #define [CR1_CLEAR_MASK](#)
- #define [CR2_CLOCK_CLEAR_MASK](#)
- #define [CR3_CLEAR_MASK](#) ((uint16_t)(USART_CR3_RTSE | USART_CR3_CTSE))
- #define [IT_MASK](#) ((uint16_t)0x001F)

Functions

- void [USART_DeInit](#) (USART_TypeDef *USARTx)
Deinitializes the USARTx peripheral registers to their default reset values.
- void [USART_Init](#) (USART_TypeDef *USARTx, [USART_InitTypeDef](#) *USART_InitStruct)
Initializes the USARTx peripheral according to the specified parameters in the USART_InitStruct .
- void [USART_StructInit](#) ([USART_InitTypeDef](#) *USART_InitStruct)
Fills each USART_InitStruct member with its default value.
- void [USART_ClockInit](#) (USART_TypeDef *USARTx, [USART_ClockInitTypeDef](#) *USART_ClockInitStruct)
Initializes the USARTx peripheral Clock according to the specified parameters in the USART_ClockInitStruct .
- void [USART_ClockStructInit](#) ([USART_ClockInitTypeDef](#) *USART_ClockInitStruct)
Fills each USART_ClockInitStruct member with its default value.
- void [USART_Cmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the specified USART peripheral.
- void [USART_SetPrescaler](#) (USART_TypeDef *USARTx, uint8_t USART_Prescaler)
Sets the system clock prescaler.
- void [USART_OverSampling8Cmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's 8x oversampling mode.
- void [USART_OneBitMethodCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's one bit sampling method.
- void [USART_SendData](#) (USART_TypeDef *USARTx, uint16_t Data)
Transmits single data through the USARTx peripheral.
- uint16_t [USART_ReceiveData](#) (USART_TypeDef *USARTx)
Returns the most recent received data by the USARTx peripheral.
- void [USART_SetAddress](#) (USART_TypeDef *USARTx, uint8_t USART_Address)
Sets the address of the USART node.

- void [USART_WakeUpConfig](#) (USART_TypeDef *USARTx, uint16_t USART_WakeUp)
Selects the USART WakeUp method.
- void [USART_ReceiverWakeUpCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Determines if the USART is in mute mode or not.
- void [USART_LINBreakDetectLengthConfig](#) (USART_TypeDef *USARTx, uint16_t USART_LINBreakDetectLength)
Sets the USART LIN Break detection length.
- void [USART_LINCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's LIN mode.
- void [USART_SendBreak](#) (USART_TypeDef *USARTx)
Transmits break characters.
- void [USART_HalfDuplexCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's Half Duplex communication.
- void [USART_SmartCardCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's Smart Card mode.
- void [USART_SmartCardNACKCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables NACK transmission.
- void [USART_SetGuardTime](#) (USART_TypeDef *USARTx, uint8_t USART_GuardTime)
Sets the specified USART guard time.
- void [USART_IrDAConfig](#) (USART_TypeDef *USARTx, uint16_t USART_IrDAMode)
Configures the USART's IrDA interface.
- void [USART_IrDACmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's IrDA interface.
- void [USART_DMAMCmd](#) (USART_TypeDef *USARTx, uint16_t USART_DMAREq, FunctionalState NewState)
Enables or disables the USART's DMA interface.
- void [USART_ITConfig](#) (USART_TypeDef *USARTx, uint16_t USART_IT, FunctionalState NewState)
Enables or disables the specified USART interrupts.
- FlagStatus [USART_GetFlagStatus](#) (USART_TypeDef *USARTx, uint16_t USART_FLAG)
Checks whether the specified USART flag is set or not.
- void [USART_ClearFlag](#) (USART_TypeDef *USARTx, uint16_t USART_FLAG)
Clears the USARTx's pending flags.
- ITStatus [USART_GetITStatus](#) (USART_TypeDef *USARTx, uint16_t USART_IT)
Checks whether the specified USART interrupt has occurred or not.
- void [USART_ClearITPendingBit](#) (USART_TypeDef *USARTx, uint16_t USART_IT)
Clears the USARTx's interrupt pending bits.

4.1.6.1 Detailed Description

USART driver modules.

4.1.6.2 Macro Definition Documentation

4.1.6.2.1 CR1_CLEAR_MASK

```
#define CR1_CLEAR_MASK
```

Value:

```
((uint16_t)(USART_CR1_M | USART_CR1_PCE | \
USART_CR1_PS | USART_CR1_TE | \
USART_CR1_RE))
```

< USART CR1 register clear Mask ((~(uint16_t)0xE9F3))

4.1.6.2.2 CR2_CLOCK_CLEAR_MASK

```
#define CR2_CLOCK_CLEAR_MASK
```

Value:

```
((uint16_t)(USART_CR2_CLKEN | USART_CR2_CPOL | \
USART_CR2_CPHA | USART_CR2_LBCL))
```

< USART CR2 register clock bits clear Mask ((~(uint16_t)0xF0FF))

4.1.6.2.3 CR3_CLEAR_MASK

```
#define CR3_CLEAR_MASK ((uint16_t)(USART_CR3_RTSE | USART_CR3_CTSE))
```

< USART CR3 register clear Mask ((~(uint16_t)0xFCFF)) USART Interrupts mask

4.1.6.2.4 IT_MASK

```
#define IT_MASK ((uint16_t)0x001F)
```

4.1.6.3 Function Documentation

4.1.6.3.1 USART_ClearFlag()

```
void USART_ClearFlag (
    USART_TypeDef * USARTx,
    uint16_t USART_FLAG )
```

Clears the USARTx's pending flags.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_FLAG</i>	specifies the flag to clear. This parameter can be any combination of the following values: <ul style="list-style-type: none"> USART_FLAG_CTS: CTS Change flag (not available for UART4 and UART5). USART_FLAG_LBD: LIN Break detection flag. USART_FLAG_TC: Transmission Complete flag. USART_FLAG_RXNE: Receive data register not empty flag.

Note

PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART_SR register ([USART_GetFlagStatus\(\)](#)) followed by a read operation to USART_DR register ([USART_ReceiveData\(\)](#)).

RXNE flag can be also cleared by a read to the USART_DR register ([USART_ReceiveData\(\)](#)).

TC flag can be also cleared by software sequence: a read operation to USART_SR register ([USART_GetFlagStatus\(\)](#)) followed by a write operation to USART_DR register ([USART_SendData\(\)](#)).

TXE flag is cleared only by a write to the USART_DR register ([USART_SendData\(\)](#)).

Return values

<i>None</i>	
-------------	--

4.1.6.3.2 USART_ClearITPendingBit()

```
void USART_ClearITPendingBit (
    USART_TypeDef * USARTx,
    uint16_t USART_IT )
```

Clears the USARTx's interrupt pending bits.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART↔ _IT</i>	specifies the interrupt pending bit to clear. This parameter can be one of the following values: <ul style="list-style-type: none"> • USART_IT_CTS: CTS change interrupt (not available for UART4 and UART5) • USART_IT_LBD: LIN Break detection interrupt • USART_IT_TC: Transmission complete interrupt. • USART_IT_RXNE: Receive Data register not empty interrupt.

Note

PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) pending bits are cleared by software sequence: a read operation to USART_SR register ([USART_GetITStatus\(\)](#)) followed by a read operation to USART_DR register ([USART_ReceiveData\(\)](#)).

RXNE pending bit can be also cleared by a read to the USART_DR register ([USART_ReceiveData\(\)](#)).

TC pending bit can be also cleared by software sequence: a read operation to USART_SR register ([USART_GetITStatus\(\)](#)) followed by a write operation to USART_DR register ([USART_SendData\(\)](#)).

TXE pending bit is cleared only by a write to the USART_DR register ([USART_SendData\(\)](#)).

Return values

<i>None</i>	
-------------	--

4.1.6.3.3 USART_ClockInit()

```
void USART_ClockInit (
    USART_TypeDef * USARTx,
    USART_ClockInitTypeDef * USART_ClockInitStruct )
```

Initializes the USARTx peripheral Clock according to the specified parameters in the USART_ClockInitStruct .

Parameters

<i>USARTx</i>	where x can be 1, 2, 3 or 6 to select the USART peripheral.
<i>USART_ClockInitStruct</i>	pointer to a USART_ClockInitTypeDef structure that contains the configuration information for the specified USART peripheral.

Note

The Smart Card and Synchronous modes are not available for UART4 and UART5.

Return values

<i>None</i>	
-------------	--

4.1.6.3.4 USART_ClockStructInit()

```
void USART_ClockStructInit (
```

```
USART_ClockInitTypeDef * USART_ClockInitStruct )
```

Fills each USART_ClockInitStruct member with its default value.

Parameters

<i>USART_ClockInitStruct</i>	pointer to a USART_ClockInitTypeDef structure which will be initialized.
------------------------------	--

Return values

<i>None</i>	
-------------	--

4.1.6.3.5 USART_Cmd()

```
void USART_Cmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the specified USART peripheral.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USARTx peripheral. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.6 USART_DeInit()

```
void USART_DeInit (
    USART_TypeDef * USARTx )
```

Deinitializes the USARTx peripheral registers to their default reset values.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
---------------	---

Return values

<i>None</i>	
-------------	--

4.1.6.3.7 USART_DMACmd()

```
void USART_DMACmd (
    USART_TypeDef * USARTx,
    uint16_t USART_DMAReq,
    FunctionalState NewState )
```

Enables or disables the USART's DMA interface.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
---------------	---

Parameters

<i>USART_DMAReq</i>	specifies the DMA request. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • USART_DMAReq_Tx: USART DMA transmit request • USART_DMAReq_Rx: USART DMA receive request
<i>NewState</i>	new state of the DMA Request sources. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.8 USART_GetFlagStatus()

```
FlagStatus USART_GetFlagStatus (
    USART_TypeDef * USARTx,
    uint16_t USART_FLAG )
```

Checks whether the specified USART flag is set or not.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_FLAG</i>	specifies the flag to check. This parameter can be one of the following values: <ul style="list-style-type: none"> • USART_FLAG_CTS: CTS Change flag (not available for UART4 and UART5) • USART_FLAG_LBD: LIN Break detection 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 values

<i>The</i>	new state of USART_FLAG (SET or RESET).
------------	---

4.1.6.3.9 USART_GetITStatus()

```
ITStatus USART_GetITStatus (
    USART_TypeDef * USARTx,
    uint16_t USART_IT )
```

Checks whether the specified USART interrupt has occurred or not.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART↔ _IT</i>	specifies the USART interrupt source to check. This parameter can be one of the following values: <ul style="list-style-type: none"> • USART_IT_CTS: CTS change interrupt (not available for UART4 and UART5) • USART_IT_LBD: LIN Break detection interrupt • 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_RX : OverRun Error interrupt if the RXNEIE bit is set • USART_IT_ORE_ER : OverRun Error interrupt if the EIE bit is set • USART_IT_NE: Noise Error interrupt • USART_IT_FE: Framing Error interrupt • USART_IT_PE: Parity Error interrupt

Return values

<i>The</i>	new state of USART_IT (SET or RESET).
------------	---------------------------------------

4.1.6.3.10 USART_HalfDuplexCmd()

```
void USART_HalfDuplexCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's Half Duplex communication.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USART Communication. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.11 USART_Init()

```
void USART_Init (
    USART_TypeDef * USARTx,
    USART_InitTypeDef * USART_InitStruct )
```

Initializes the USARTx peripheral according to the specified parameters in the USART_InitStruct .

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_InitStruct</i>	pointer to a USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.

Return values

<i>None</i>	
-------------	--

4.1.6.3.12 USART_IrDACmd()

```
void USART_IrDACmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's IrDA interface.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the IrDA mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.13 USART_IrDAConfig()

```
void USART_IrDAConfig (
    USART_TypeDef * USARTx,
    uint16_t USART_IrDAMode )
```

Configures the USART's IrDA interface.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_IrDAMode</i>	specifies the IrDA mode. This parameter can be one of the following values: <ul style="list-style-type: none"> USART_IrDAMode_LowPower USART_IrDAMode_Normal

Return values

<i>None</i>	
-------------	--

4.1.6.3.14 USART_ITConfig()

```
void USART_ITConfig (
    USART_TypeDef * USARTx,
    uint16_t USART_IT,
```

```
FunctionalState NewState )
```

Enables or disables the specified USART interrupts.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_↔_IT</i>	specifies the USART interrupt sources to be enabled or disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> • USART_IT_CTS: CTS change interrupt • USART_IT_LBD: LIN Break detection interrupt • 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)
<i>NewState</i>	new state of the specified USARTx interrupts. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.15 USART_LINBreakDetectLengthConfig()

```
void USART_LINBreakDetectLengthConfig (
    USART_TypeDef * USARTx,
    uint16_t USART_LINBreakDetectLength )
```

Sets the USART LIN Break detection length.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_LINBreakDetectLength</i>	specifies the LIN break detection length. This parameter can be one of the following values: <ul style="list-style-type: none"> • USART_LINBreakDetectLength_10b: 10-bit break detection • USART_LINBreakDetectLength_11b: 11-bit break detection

Return values

<i>None</i>	
-------------	--

4.1.6.3.16 USART_LINCmd()

```
void USART_LINCmd (
    USART_TypeDef * USARTx,
```

```
FunctionalState NewState )
```

Enables or disables the USART's LIN mode.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USART LIN mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.17 USART_OneBitMethodCmd()

```
void USART_OneBitMethodCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's one bit sampling method.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USART one bit sampling method. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.18 USART_OverSampling8Cmd()

```
void USART_OverSampling8Cmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's 8x oversampling mode.

Note

This function has to be called before calling [USART_Init\(\)](#) function in order to have correct baudrate Divider value.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USART 8x oversampling mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.19 USART_ReceiveData()

```
uint16_t USART_ReceiveData (
    USART_TypeDef * USARTx )
```

Returns the most recent received data by the USARTx peripheral.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
---------------	---

Return values

<i>The</i>	received data.
------------	----------------

4.1.6.3.20 USART_ReceiverWakeUpCmd()

```
void USART_ReceiverWakeUpCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Determines if the USART is in mute mode or not.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USART mute mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.21 USART_SendBreak()

```
void USART_SendBreak (
    USART_TypeDef * USARTx )
```

Transmits break characters.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
---------------	---

Return values

<i>None</i>	
-------------	--

4.1.6.3.22 USART_SendData()

```
void USART_SendData (
    USART_TypeDef * USARTx,
    uint16_t Data )
```

Transmits single data through the USARTx peripheral.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>Data</i>	the data to transmit.

Return values

<i>None</i>	
-------------	--

4.1.6.3.23 USART_SetAddress()

```
void USART_SetAddress (
    USART_TypeDef * USARTx,
    uint8_t USART_Address )
```

Sets the address of the USART node.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_Address</i>	Indicates the address of the USART node.

Return values

<i>None</i>	
-------------	--

4.1.6.3.24 USART_SetGuardTime()

```
void USART_SetGuardTime (
    USART_TypeDef * USARTx,
    uint8_t USART_GuardTime )
```

Sets the specified USART guard time.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3 or 6 to select the USART or UART peripheral.
<i>USART_GuardTime</i>	specifies the guard time.

Return values

<i>None</i>	
-------------	--

4.1.6.3.25 USART_SetPrescaler()

```
void USART_SetPrescaler (
    USART_TypeDef * USARTx,
    uint8_t USART_Prescaler )
```

Sets the system clock prescaler.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
---------------	---

Parameters

<i>USART_Prescaler</i>	specifies the prescaler clock.
------------------------	--------------------------------

Note

The function is used for IrDA mode with UART4 and UART5.

Return values

<i>None</i>	
-------------	--

4.1.6.3.26 USART_SmartCardCmd()

```
void USART_SmartCardCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's Smart Card mode.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the Smart Card mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.27 USART_SmartCardNACKCmd()

```
void USART_SmartCardNACKCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables NACK transmission.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the NACK transmission. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.3.28 USART_StructInit()

```
void USART_StructInit (
    USART_InitTypeDef * USART_InitStruct )
```

Fills each USART_InitStruct member with its default value.

Parameters

<code>USART_InitStruct</code>	pointer to a USART_InitTypeDef structure which will be initialized.
-------------------------------	---

Return values

<code>None</code>	
-------------------	--

4.1.6.3.29 USART_WakeUpConfig()

```
void USART_WakeUpConfig (
    USART_TypeDef * USARTx,
    uint16_t USART_WakeUp )
```

Selects the USART WakeUp method.

Parameters

<code>USARTx</code>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<code>USART_WakeUp</code>	specifies the USART wakeup method. This parameter can be one of the following values: <ul style="list-style-type: none"> • <code>USART_WakeUp_IdleLine</code>: WakeUp by an idle line detection • <code>USART_WakeUp_AddressMark</code>: WakeUp by an address mark

Return values

<code>None</code>	
-------------------	--

4.1.6.4 USART_Private_Functions**Modules**

- [Initialization and Configuration functions](#)
Initialization and Configuration functions.
- [Data transfers functions](#)
Data transfers functions.
- [MultiProcessor Communication functions](#)
Multi-Processor Communication functions.
- [LIN mode functions](#)
LIN mode functions.
- [Halfduplex mode function](#)
Half-duplex mode function.
- [Smartcard mode functions](#)
Smartcard mode functions.
- [IrDA mode functions](#)
IrDA mode functions.
- [DMA transfers management functions](#)
DMA transfers management functions.
- [Interrupts and flags management functions](#)
Interrupts and flags management functions.

4.1.6.4.1 Detailed Description

4.1.6.4.2 Initialization and Configuration functions

Initialization and Configuration functions.

Functions

- void **USART_DeInit** (USART_TypeDef *USARTx)
Deinitializes the USARTx peripheral registers to their default reset values.
- void **USART_Init** (USART_TypeDef *USARTx, **USART_InitTypeDef** *USART_InitStruct)
Initializes the USARTx peripheral according to the specified parameters in the USART_InitStruct .
- void **USART_StructInit** (**USART_InitTypeDef** *USART_InitStruct)
Fills each USART_InitStruct member with its default value.
- void **USART_ClockInit** (USART_TypeDef *USARTx, **USART_ClockInitTypeDef** *USART_ClockInitStruct)
Initializes the USARTx peripheral Clock according to the specified parameters in the USART_ClockInitStruct .
- void **USART_ClockStructInit** (**USART_ClockInitTypeDef** *USART_ClockInitStruct)
Fills each USART_ClockInitStruct member with its default value.
- void **USART_Cmd** (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the specified USART peripheral.
- void **USART_SetPrescaler** (USART_TypeDef *USARTx, uint8_t USART_Prescaler)
Sets the system clock prescaler.
- void **USART_OverSampling8Cmd** (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's 8x oversampling mode.
- void **USART_OneBitMethodCmd** (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's one bit sampling method.

4.1.6.4.2.1 Detailed Description

Initialization and Configuration functions.

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:

M bit		PCE bit		USART frame		
0	0			SB	8 bit data	STB
0	1			SB	7 bit data	PB STB
1	0			SB	9 bit data	STB
1	1			SB	8 bit data	PB STB

- Hardware flow control
- Receiver/transmitter modes

The USART_Init() function follows the USART asynchronous configuration procedure (details for the procedure are available in reference manual (RM0090)).

- For the synchronous mode in addition to the asynchronous mode parameters these parameters should be also configured:

- USART Clock Enabled
- USART polarity
- USART phase
- USART LastBit

These parameters can be configured using the `USART_ClockInit()` function.

4.1.6.4.2.2 Function Documentation

USART_ClockInit()

```
void USART_ClockInit (
    USART_TypeDef * USARTx,
    USART_ClockInitTypeDef * USART_ClockInitStruct )
```

Initializes the USARTx peripheral Clock according to the specified parameters in the `USART_ClockInitStruct`.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3 or 6 to select the USART peripheral.
<i>USART_ClockInitStruct</i>	pointer to a USART_ClockInitTypeDef structure that contains the configuration information for the specified USART peripheral.

Note

The Smart Card and Synchronous modes are not available for UART4 and UART5.

Return values

<i>None</i>	
-------------	--

USART_ClockStructInit()

```
void USART_ClockStructInit (
    USART_ClockInitTypeDef * USART_ClockInitStruct )
```

Fills each `USART_ClockInitStruct` member with its default value.

Parameters

<i>USART_ClockInitStruct</i>	pointer to a USART_ClockInitTypeDef structure which will be initialized.
------------------------------	--

Return values

<i>None</i>	
-------------	--

USART_Cmd()

```
void USART_Cmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the specified USART peripheral.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USARTx peripheral. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

USART_DeInit()

```
void USART_DeInit (
    USART_TypeDef * USARTx )
```

Deinitializes the USARTx peripheral registers to their default reset values.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
---------------	---

Return values

<i>None</i>	
-------------	--

USART_Init()

```
void USART_Init (
    USART_TypeDef * USARTx,
    USART_InitTypeDef * USART_InitStruct )
```

Initializes the USARTx peripheral according to the specified parameters in the USART_InitStruct .

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_InitStruct</i>	pointer to a USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.

Return values

<i>None</i>	
-------------	--

USART_OneBitMethodCmd()

```
void USART_OneBitMethodCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's one bit sampling method.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USART one bit sampling method. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

USART_OverSampling8Cmd()

```
void USART_OverSampling8Cmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's 8x oversampling mode.

Note

This function has to be called before calling [USART_Init\(\)](#) function in order to have correct baudrate Divider value.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USART 8x oversampling mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

USART_SetPrescaler()

```
void USART_SetPrescaler (
    USART_TypeDef * USARTx,
    uint8_t USART_Prescaler )
```

Sets the system clock prescaler.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_Prescaler</i>	specifies the prescaler clock.

Note

The function is used for IrDA mode with UART4 and UART5.

Return values

<i>None</i>	
-------------	--

USART_StructInit()

```
void USART_StructInit (
    USART_InitTypeDef * USART_InitStruct )
```

Fills each USART_InitStruct member with its default value.

Parameters

<i>USART_InitStruct</i>	pointer to a USART_InitTypeDef structure which will be initialized.
-------------------------	---

Return values

<i>None</i>	
-------------	--

4.1.6.4.3 Data transfers functions

Data transfers functions.

Functions

- void [USART_SendData](#) (USART_TypeDef *USARTx, uint16_t Data)
Transmits single data through the USARTx peripheral.
- uint16_t [USART_ReceiveData](#) (USART_TypeDef *USARTx)
Returns the most recent received data by the USARTx peripheral.

4.1.6.4.3.1 Detailed Description

Data transfers functions.

```
=====
                        Data transfers functions
=====
```

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

During an USART reception, data shifts in least significant bit first through the RX pin. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register.

When a transmission is taking place, a write instruction to the USART_DR register stores the data in the TDR register and which is copied in the shift register at the end of the current transmission.

The read access of the USART_DR register can be done using the USART_ReceiveData() function and returns the RDR buffered value. Whereas a write access to the USART_DR can be done using USART_SendData() function and stores the written data into TDR buffer.

4.1.6.4.3.2 Function Documentation

USART_ReceiveData()

```
uint16_t USART_ReceiveData (
    USART_TypeDef * USARTx )
```

Returns the most recent received data by the USARTx peripheral.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
---------------	---

Return values

<i>The</i>	received data.
------------	----------------

USART_SendData()

```
void USART_SendData (
    USART_TypeDef * USARTx,
    uint16_t Data )
```

Transmits single data through the USARTx peripheral.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>Data</i>	the data to transmit.

Return values

<i>None</i>	
-------------	--

4.1.6.4.4 MultiProcessor Communication functions

Multi-Processor Communication functions.

Functions

- void [USART_SetAddress](#) (USART_TypeDef *USARTx, uint8_t USART_Address)
Sets the address of the USART node.
- void [USART_ReceiverWakeUpCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Determines if the USART is in mute mode or not.
- void [USART_WakeUpConfig](#) (USART_TypeDef *USARTx, uint16_t USART_WakeUp)
Selects the USART WakeUp method.

4.1.6.4.4.1 Detailed Description

Multi-Processor Communication functions.

```
=====
                        Multi-Processor Communication functions
=====
```

This subsection provides a set of functions allowing to manage the USART multiprocessor communication.

For instance one of the USARTs can be the master, its TX output is connected to the RX input of the other USART. The others are slaves, their respective TX outputs are logically ANDed together and connected to the RX input of the master.

USART multiprocessor communication is possible through the following procedure:

1. Program the Baud rate, Word length = 9 bits, Stop bits, Parity, Mode transmitter or Mode receiver and hardware flow control values using the USART_Init() function.
2. Configures the USART address using the USART_SetAddress() function.
3. Configures the wake up method (USART_WakeUp_IdleLine or USART_WakeUp_AddressMark) using USART_WakeUpConfig() function only for the slaves.
4. Enable the USART using the USART_Cmd() function.
5. Enter the USART slaves in mute mode using USART_ReceiverWakeUpCmd() function.

The USART Slave exit from mute mode when receive the wake up condition.

4.1.6.4.4.2 Function Documentation

USART_ReceiverWakeUpCmd()

```
void USART_ReceiverWakeUpCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Determines if the USART is in mute mode or not.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USART mute mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

USART_SetAddress()

```
void USART_SetAddress (
    USART_TypeDef * USARTx,
    uint8_t USART_Address )
```

Sets the address of the USART node.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_Address</i>	Indicates the address of the USART node.

Return values

<i>None</i>	
-------------	--

USART_WakeUpConfig()

```
void USART_WakeUpConfig (
    USART_TypeDef * USARTx,
    uint16_t USART_WakeUp )
```

Selects the USART WakeUp method.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_WakeUp</i>	specifies the USART wakeup method. This parameter can be one of the following values: <ul style="list-style-type: none"> USART_WakeUp_IdleLine: WakeUp by an idle line detection USART_WakeUp_AddressMark: WakeUp by an address mark

Return values

<i>None</i>	
-------------	--

4.1.6.4.5 LIN mode functions

LIN mode functions.

Functions

- void [USART_LINBreakDetectLengthConfig](#) (USART_TypeDef *USARTx, uint16_t USART_LINBreakDetectLength)
Sets the USART LIN Break detection length.
- void [USART_LINCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's LIN mode.
- void [USART_SendBreak](#) (USART_TypeDef *USARTx)
Transmits break characters.

4.1.6.4.5.1 Detailed Description

LIN mode functions.

```
=====
                        LIN mode functions
=====
```

This subsection provides a set of functions allowing to manage the USART LIN Mode communication.

In LIN mode, 8-bit data format with 1 stop bit is required in accordance with the LIN standard.

Only this LIN Feature is supported by the USART IP:

- LIN Master Synchronous Break send capability and LIN slave break detection capability : 13-bit break generation and 10/11 bit break detection

USART LIN Master transmitter communication is possible through the following procedure:

1. Program the Baud rate, Word length = 8bits, Stop bits = 1bit, Parity, Mode transmitter or Mode receiver and hardware flow control values using the USART_Init() function.
2. Enable the USART using the USART_Cmd() function.
3. Enable the LIN mode using the USART_LINCmd() function.
4. Send the break character using USART_SendBreak() function.

USART LIN Master receiver communication is possible through the following procedure:

1. Program the Baud rate, Word length = 8bits, Stop bits = 1bit, Parity, Mode transmitter or Mode receiver and hardware flow control values using the USART_Init() function.
2. Enable the USART using the USART_Cmd() function.
3. Configures the break detection length using the USART_LINBreakDetectLengthConfig() function.
4. Enable the LIN mode using the USART_LINCmd() function.

@note In LIN mode, the following bits must be kept cleared:

- CLKEN in the USART_CR2 register,
- STOP[1:0], SCEN, HDSEL and IREN in the USART_CR3 register.

4.1.6.4.5.2 Function Documentation

USART_LINBreakDetectLengthConfig()

```
void USART_LINBreakDetectLengthConfig (
    USART_TypeDef * USARTx,
    uint16_t USART_LINBreakDetectLength )
```

Sets the USART LIN Break detection length.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
---------------	---

Parameters

<i>USART_LINBreakDetectLength</i>	specifies the LIN break detection length. This parameter can be one of the following values: <ul style="list-style-type: none"> • USART_LINBreakDetectLength_10b: 10-bit break detection • USART_LINBreakDetectLength_11b: 11-bit break detection
-----------------------------------	---

Return values

<i>None</i>	
-------------	--

USART_LINCmd()

```
void USART_LINCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's LIN mode.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USART LIN mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

USART_SendBreak()

```
void USART_SendBreak (
    USART_TypeDef * USARTx )
```

Transmits break characters.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
---------------	---

Return values

<i>None</i>	
-------------	--

4.1.6.4.6 Halfduplex mode function

Half-duplex mode function.

Functions

- void [USART_HalfDuplexCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's Half Duplex communication.

4.1.6.4.6.1 Detailed Description

Half-duplex mode function.

```
=====
                        Half-duplex mode function
=====
```

This subsection provides a set of functions allowing to manage the USART Half-duplex communication.

The USART can be configured to follow a single-wire half-duplex protocol where the TX and RX lines are internally connected.

USART Half duplex communication is possible through the following procedure:

1. Program the Baud rate, Word length, Stop bits, Parity, Mode transmitter or Mode receiver and hardware flow control values using the USART_Init() function.
2. Configures the USART address using the USART_SetAddress() function.
3. Enable the USART using the USART_Cmd() function.
4. Enable the half duplex mode using USART_HalfDuplexCmd() function.

@note The RX pin is no longer used

@note In Half-duplex mode the following bits must be kept cleared:

- LINEN and CLKEN bits in the USART_CR2 register.
- SCEN and IREN bits in the USART_CR3 register.

4.1.6.4.6.2 Function Documentation

USART_HalfDuplexCmd()

```
void USART_HalfDuplexCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's Half Duplex communication.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the USART Communication. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.4.7 Smartcard mode functions

Smartcard mode functions.

Functions

- void [USART_SetGuardTime](#) (USART_TypeDef *USARTx, uint8_t USART_GuardTime)
Sets the specified USART guard time.
- void [USART_SmartCardCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's Smart Card mode.
- void [USART_SmartCardNACKCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables NACK transmission.

4.1.6.4.7.1 Detailed Description

Smartcard mode functions.

 Smartcard mode functions

This subsection provides a set of functions allowing to manage the USART Smartcard communication.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard.

The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler.

Smartcard communication is possible through the following procedure:

1. Configures the Smartcard Prescaler using the USART_SetPrescaler() function.
2. Configures the Smartcard Guard Time using the USART_SetGuardTime() function.
3. Program the USART clock using the USART_ClockInit() function as following:
 - USART Clock enabled
 - USART CPOL Low
 - USART CPHA on first edge
 - USART Last Bit Clock Enabled
4. Program the Smartcard interface using the USART_Init() function as following:
 - Word Length = 9 Bits
 - 1.5 Stop Bit
 - Even parity
 - BaudRate = 12096 baud
 - Hardware flow control disabled (RTS and CTS signals)
 - Tx and Rx enabled
5. Optionally you can enable the parity error interrupt using the USART_ITConfig() function
6. Enable the USART using the USART_Cmd() function.
7. Enable the Smartcard NACK using the USART_SmartCardNACKCmd() function.
8. Enable the Smartcard interface using the USART_SmartCardCmd() function.

Please refer to the ISO 7816-3 specification for more details.

@note It is also possible to choose 0.5 stop bit for receiving but it is recommended to use 1.5 stop bits for both transmitting and receiving to avoid switching between the two configurations.

@note In smartcard mode, the following bits must be kept cleared:

- LINEN bit in the USART_CR2 register.
- HDSEL and IREN bits in the USART_CR3 register.

@note Smartcard mode is available on USART peripherals only (not available on UART4 and UART5 peripherals).

4.1.6.4.7.2 Function Documentation

USART_SetGuardTime()

```
void USART_SetGuardTime (
    USART_TypeDef * USARTx,
    uint8_t USART_GuardTime )
```

Sets the specified USART guard time.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3 or 6 to select the USART or UART peripheral.
<i>USART_GuardTime</i>	specifies the guard time.

Return values

<i>None</i>	
-------------	--

USART_SmartCardCmd()

```
void USART_SmartCardCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's Smart Card mode.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the Smart Card mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

USART_SmartCardNACKCmd()

```
void USART_SmartCardNACKCmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables NACK transmission.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the NACK transmission. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.4.8 IrDA mode functions

IrDA mode functions.

Functions

- void [USART_IrDAConfig](#) (USART_TypeDef *USARTx, uint16_t USART_IrDAMode)
Configures the USART's IrDA interface.
- void [USART_IrDACmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's IrDA interface.

4.1.6.4.8.1 Detailed Description

IrDA mode functions.

```
=====
                                IrDA mode functions
=====
```

This subsection provides a set of functions allowing to manage the USART IrDA communication.

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

IrDA communication is possible through the following procedure:

1. Program the Baud rate, Word length = 8 bits, Stop bits, Parity, Transmitter/Receiver modes and hardware flow control values using the USART_Init() function.
2. Enable the USART using the USART_Cmd() function.
3. Configures the IrDA pulse width by configuring the prescaler using the USART_SetPrescaler() function.
4. Configures the IrDA USART_IrDAMode_LowPower or USART_IrDAMode_Normal mode using the USART_IrDAConfig() function.
5. Enable the IrDA using the USART_IrDACmd() function.

@note A pulse of width less than two and greater than one PSC period(s) may or may not be rejected.

@note The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).

@note In IrDA mode, the following bits must be kept cleared:

- LINEN, STOP and CLKEN bits in the USART_CR2 register.
- SCEN and HDSEL bits in the USART_CR3 register.

4.1.6.4.8.2 Function Documentation

USART_IrDACmd()

```
void USART_IrDACmd (
    USART_TypeDef * USARTx,
    FunctionalState NewState )
```

Enables or disables the USART's IrDA interface.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>NewState</i>	new state of the IrDA mode. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

USART_IrDAConfig()

```
void USART_IrDAConfig (
    USART_TypeDef * USARTx,
    uint16_t USART_IrDAMode )
```

Configures the USART's IrDA interface.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_IrDAMode</i>	specifies the IrDA mode. This parameter can be one of the following values: <ul style="list-style-type: none"> • USART_IrDAMode_LowPower • USART_IrDAMode_Normal

Return values

None	
------	--

4.1.6.4.9 DMA transfers management functions

DMA transfers management functions.

Functions

- void [USART_DMACmd](#) (USART_TypeDef *USARTx, uint16_t USART_DMAREq, FunctionalState NewState)
Enables or disables the USART's DMA interface.

4.1.6.4.9.1 Detailed Description

DMA transfers management functions.

```
=====
DMA transfers management functions
=====
```

4.1.6.4.9.2 Function Documentation**USART_DMACmd()**

```
void USART_DMACmd (
    USART_TypeDef * USARTx,
    uint16_t USART_DMAREq,
    FunctionalState NewState )
```

Enables or disables the USART's DMA interface.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_DMAREq</i>	specifies the DMA request. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • USART_DMAREq_Tx: USART DMA transmit request • USART_DMAREq_Rx: USART DMA receive request
<i>NewState</i>	new state of the DMA Request sources. This parameter can be: ENABLE or DISABLE.

Return values

None	
------	--

4.1.6.4.10 Interrupts and flags management functions

Interrupts and flags management functions.

Functions

- void [USART_ITConfig](#) (USART_TypeDef *USARTx, uint16_t USART_IT, FunctionalState NewState)
Enables or disables the specified USART interrupts.
- FlagStatus [USART_GetFlagStatus](#) (USART_TypeDef *USARTx, uint16_t USART_FLAG)
Checks whether the specified USART flag is set or not.

- void **USART_ClearFlag** (USART_TypeDef *USARTx, uint16_t USART_FLAG)
Clears the USARTx's pending flags.
- ITStatus **USART_GetITStatus** (USART_TypeDef *USARTx, uint16_t USART_IT)
Checks whether the specified USART interrupt has occurred or not.
- void **USART_ClearITPendingBit** (USART_TypeDef *USARTx, uint16_t USART_IT)
Clears the USARTx's interrupt pending bits.

4.1.6.4.10.1 Detailed Description

Interrupts and flags management functions.

```
=====
                        Interrupts and flags management functions
=====
```

This subsection provides a set of functions allowing to configure the USART Interrupts sources, DMA channels requests and check or clear the flags or pending bits status.
The user should identify which mode will be used in his application to manage the communication: Polling mode, Interrupt mode or DMA mode.

Polling Mode

=====

In Polling Mode, the SPI communication can be managed by 10 flags:

1. USART_FLAG_TXE : to indicate the status of the transmit buffer register
2. USART_FLAG_RXNE : to indicate the status of the receive buffer register
3. USART_FLAG_TC : to indicate the status of the transmit operation
4. USART_FLAG_IDLE : to indicate the status of the Idle Line
5. USART_FLAG_CTS : to indicate the status of the nCTS input
6. USART_FLAG_LBD : to indicate the status of the LIN break detection
7. USART_FLAG_NE : to indicate if a noise error occur
8. USART_FLAG_FE : to indicate if a frame error occur
9. USART_FLAG_PE : to indicate if a parity error occur
10. USART_FLAG_ORE : to indicate if an Overrun error occur

In this Mode it is advised to use the following functions:

- FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG);
- void USART_ClearFlag(USART_TypeDef* USARTx, uint16_t USART_FLAG);

Interrupt Mode

=====

In Interrupt Mode, the USART communication can be managed by 8 interrupt sources and 10 pending bits:

Pending Bits:

1. USART_IT_TXE : to indicate the status of the transmit buffer register
2. USART_IT_RXNE : to indicate the status of the receive buffer register
3. USART_IT_TC : to indicate the status of the transmit operation
4. USART_IT_IDLE : to indicate the status of the Idle Line
5. USART_IT_CTS : to indicate the status of the nCTS input
6. USART_IT_LBD : to indicate the status of the LIN break detection
7. USART_IT_NE : to indicate if a noise error occur
8. USART_IT_FE : to indicate if a frame error occur
9. USART_IT_PE : to indicate if a parity error occur
10. USART_IT_ORE : to indicate if an Overrun error occur

Interrupt Source:

1. USART_IT_TXE : specifies the interrupt source for the Tx buffer empty interrupt.
2. USART_IT_RXNE : specifies the interrupt source for the Rx buffer not empty interrupt.
3. USART_IT_TC : specifies the interrupt source for the Transmit complete interrupt.
4. USART_IT_IDLE : specifies the interrupt source for the Idle Line interrupt.
5. USART_IT_CTS : specifies the interrupt source for the CTS interrupt.
6. USART_IT_LBD : specifies the interrupt source for the LIN break detection interrupt.
7. USART_IT_PE : specifies the interrupt source for the parity error interrupt.

8. USART_IT_ERR : specifies the interrupt source for the errors interrupt.

@note Some parameters are coded in order to use them as interrupt source or as pending bits.

In this Mode it is advised to use the following functions:

- void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT, FunctionalState NewState);
- ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT);
- void USART_ClearITPendingBit(USART_TypeDef* USARTx, uint16_t USART_IT);

DMA Mode
=====

In DMA Mode, the USART communication can be managed by 2 DMA Channel requests:

1. USART_DMAREq_Tx: specifies the Tx buffer DMA transfer request
2. USART_DMAREq_Rx: specifies the Rx buffer DMA transfer request

In this Mode it is advised to use the following function:

- void USART_DMACmd(USART_TypeDef* USARTx, uint16_t USART_DMAREq, FunctionalState NewState);

4.1.6.4.10.2 Function Documentation

USART_ClearFlag()

```
void USART_ClearFlag (
    USART_TypeDef * USARTx,
    uint16_t USART_FLAG )
```

Clears the USARTx's pending flags.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_FLAG</i>	specifies the flag to clear. This parameter can be any combination of the following values: <ul style="list-style-type: none"> • USART_FLAG_CTS: CTS Change flag (not available for UART4 and UART5). • USART_FLAG_LBD: LIN Break detection flag. • USART_FLAG_TC: Transmission Complete flag. • USART_FLAG_RXNE: Receive data register not empty flag.

Note

PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART_SR register ([USART_GetFlagStatus\(\)](#)) followed by a read operation to USART_DR register ([USART_ReceiveData\(\)](#)).

RXNE flag can be also cleared by a read to the USART_DR register ([USART_ReceiveData\(\)](#)).

TC flag can be also cleared by software sequence: a read operation to USART_SR register ([USART_GetFlagStatus\(\)](#)) followed by a write operation to USART_DR register ([USART_SendData\(\)](#)).

TXE flag is cleared only by a write to the USART_DR register ([USART_SendData\(\)](#)).

Return values

<i>None</i>	
-------------	--

USART_ClearITPendingBit()

```
void USART_ClearITPendingBit (
    USART_TypeDef * USARTx,
    uint16_t USART_IT )
```

Clears the USARTx's interrupt pending bits.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_↔_IT</i>	specifies the interrupt pending bit to clear. This parameter can be one of the following values: <ul style="list-style-type: none"> • USART_IT_CTS: CTS change interrupt (not available for UART4 and UART5) • USART_IT_LBD: LIN Break detection interrupt • USART_IT_TC: Transmission complete interrupt. • USART_IT_RXNE: Receive Data register not empty interrupt.

Note

PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) pending bits are cleared by software sequence: a read operation to USART_SR register ([USART_GetITStatus\(\)](#)) followed by a read operation to USART_DR register ([USART_ReceiveData\(\)](#)).

RXNE pending bit can be also cleared by a read to the USART_DR register ([USART_ReceiveData\(\)](#)).

TC pending bit can be also cleared by software sequence: a read operation to USART_SR register ([USART_GetITStatus\(\)](#)) followed by a write operation to USART_DR register ([USART_SendData\(\)](#)).

TXE pending bit is cleared only by a write to the USART_DR register ([USART_SendData\(\)](#)).

Return values

<i>None</i>	
-------------	--

USART_GetFlagStatus()

```
FlagStatus USART_GetFlagStatus (
    USART_TypeDef * USARTx,
    uint16_t USART_FLAG )
```

Checks whether the specified USART flag is set or not.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_FLAG</i>	specifies the flag to check. This parameter can be one of the following values: <ul style="list-style-type: none"> • USART_FLAG_CTS: CTS Change flag (not available for UART4 and UART5) • USART_FLAG_LBD: LIN Break detection 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 values

<i>The</i>	new state of USART_FLAG (SET or RESET).
------------	---

USART_GetITStatus()

```
ITStatus USART_GetITStatus (
    USART_TypeDef * USARTx,
    uint16_t USART_IT )
```

Checks whether the specified USART interrupt has occurred or not.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART↔ _IT</i>	<p>specifies the USART interrupt source to check. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • USART_IT_CTS: CTS change interrupt (not available for UART4 and UART5) • USART_IT_LBD: LIN Break detection interrupt • 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_RX : OverRun Error interrupt if the RXNEIE bit is set • USART_IT_ORE_ER : OverRun Error interrupt if the EIE bit is set • USART_IT_NE: Noise Error interrupt • USART_IT_FE: Framing Error interrupt • USART_IT_PE: Parity Error interrupt

Return values

<i>The</i>	new state of USART_IT (SET or RESET).
------------	---------------------------------------

USART_ITConfig()

```
void USART_ITConfig (
    USART_TypeDef * USARTx,
    uint16_t USART_IT,
    FunctionalState NewState )
```

Enables or disables the specified USART interrupts.

Parameters

<i>USARTx</i>	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
---------------	---

Parameters

<i>USART↔ _IT</i>	<p>specifies the USART interrupt sources to be enabled or disabled. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • USART_IT_CTS: CTS change interrupt • USART_IT_LBD: LIN Break detection interrupt • 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)
<i>NewState</i>	new state of the specified USARTx interrupts. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>	
-------------	--

4.1.6.5 USART_Exported_Constants

Modules

- [USART_Word_Length](#)
- [USART_Stop_Bits](#)
- [USART_Parity](#)
- [USART_Mode](#)
- [USART_Hardware_Flow_Control](#)
- [USART_Clock](#)
- [USART_Clock_Polarity](#)
- [USART_Clock_Phase](#)
- [USART_Last_Bit](#)
- [USART_Interrupt_definition](#)
- [USART_DMA_Requests](#)
- [USART_WakeUp_methods](#)
- [USART_LIN_Break_Detection_Length](#)
- [USART_IrDA_Low_Power](#)
- [USART_Flags](#)

Macros

- `#define IS_USART_ALL_PERIPH(PERIPH)`
- `#define IS_USART_1236_PERIPH(PERIPH)`

4.1.6.5.1 Detailed Description

4.1.6.5.2 Macro Definition Documentation

4.1.6.5.2.1 IS_USART_1236_PERIPH

```
#define IS_USART_1236_PERIPH(  
    PERIPH )
```

Value:

```
(( (PERIPH) == USART1) || \
 (PERIPH) == USART2) || \
 (PERIPH) == USART3) || \
 (PERIPH) == USART6)
```

4.1.6.5.2.2 IS_USART_ALL_PERIPH

```
#define IS_USART_ALL_PERIPH(  
    PERIPH )
```

Value:

```
(( (PERIPH) == USART1) || \
 (PERIPH) == USART2) || \
 (PERIPH) == USART3) || \
 (PERIPH) == UART4) || \
 (PERIPH) == UART5) || \
 (PERIPH) == USART6)
```

4.1.6.5.3 USART_Word_Length

Macros

- #define [USART_WordLength_8b](#) ((uint16_t)0x0000)
- #define [USART_WordLength_9b](#) ((uint16_t)0x1000)
- #define [IS_USART_WORD_LENGTH](#)(LENGTH)

4.1.6.5.3.1 Detailed Description

4.1.6.5.3.2 Macro Definition Documentation

IS_USART_WORD_LENGTH

```
#define IS_USART_WORD_LENGTH(  
    LENGTH )
```

Value:

```
(( (LENGTH) == USART\_WordLength\_8b) || \
 (LENGTH) == USART\_WordLength\_9b)
```

USART_WordLength_8b

```
#define USART_WordLength_8b ((uint16_t)0x0000)
```

USART_WordLength_9b

```
#define USART_WordLength_9b ((uint16_t)0x1000)
```

4.1.6.5.4 USART_Stop_Bits

Macros

- #define [USART_StopBits_1](#) ((uint16_t)0x0000)
- #define [USART_StopBits_0_5](#) ((uint16_t)0x1000)
- #define [USART_StopBits_2](#) ((uint16_t)0x2000)
- #define [USART_StopBits_1_5](#) ((uint16_t)0x3000)
- #define [IS_USART_STOPBITS](#)(STOPBITS)

4.1.6.5.4.1 Detailed Description

4.1.6.5.4.2 Macro Definition Documentation

IS_USART_STOPBITS

```
#define IS_USART_STOPBITS(  
    STOPBITS )
```

Value:

```
(( (STOPBITS) == USART\_StopBits\_1) || \
```



```
((STOPBITS) == USART_StopBits_0_5) || \
((STOPBITS) == USART_StopBits_2) || \
((STOPBITS) == USART_StopBits_1_5))
```

USART_StopBits_0_5

```
#define USART_StopBits_0_5 ((uint16_t)0x1000)
```

USART_StopBits_1

```
#define USART_StopBits_1 ((uint16_t)0x0000)
```

USART_StopBits_1_5

```
#define USART_StopBits_1_5 ((uint16_t)0x3000)
```

USART_StopBits_2

```
#define USART_StopBits_2 ((uint16_t)0x2000)
```

4.1.6.5.5 USART_Parity

Macros

- #define `USART_Parity_No` ((uint16_t)0x0000)
- #define `USART_Parity_Even` ((uint16_t)0x0400)
- #define `USART_Parity_Odd` ((uint16_t)0x0600)
- #define `IS_USART_PARITY`(PARITY)

4.1.6.5.5.1 Detailed Description

4.1.6.5.5.2 Macro Definition Documentation

IS_USART_PARITY

```
#define IS_USART_PARITY(  
    PARITY )
```

Value:

```
((PARITY) == USART_Parity_No) || \
((PARITY) == USART_Parity_Even) || \
((PARITY) == USART_Parity_Odd)
```

USART_Parity_Even

```
#define USART_Parity_Even ((uint16_t)0x0400)
```

USART_Parity_No

```
#define USART_Parity_No ((uint16_t)0x0000)
```

USART_Parity_Odd

```
#define USART_Parity_Odd ((uint16_t)0x0600)
```

4.1.6.5.6 USART_Mode

Macros

- #define `USART_Mode_Rx` ((uint16_t)0x0004)
- #define `USART_Mode_Tx` ((uint16_t)0x0008)
- #define `IS_USART_MODE`(MODE) (((MODE) & (uint16_t)0xFFFF3) == 0x00) && ((MODE) != (uint16_t)0x00)

4.1.6.5.6.1 Detailed Description

4.1.6.5.6.2 Macro Definition Documentation

IS_USART_MODE

```
#define IS_USART_MODE(  
    MODE ) (((MODE) & (uint16_t)0xFFF3) == 0x00) && ((MODE) != (uint16_t)0x00)
```

USART_Mode_Rx

```
#define USART_Mode_Rx ((uint16_t)0x0004)
```

USART_Mode_Tx

```
#define USART_Mode_Tx ((uint16_t)0x0008)
```

4.1.6.5.7 USART_Hardware_Flow_Control

Macros

- `#define USART_HardwareFlowControl_None ((uint16_t)0x0000)`
- `#define USART_HardwareFlowControl_RTS ((uint16_t)0x0100)`
- `#define USART_HardwareFlowControl_CTS ((uint16_t)0x0200)`
- `#define USART_HardwareFlowControl_RTS_CTS ((uint16_t)0x0300)`
- `#define IS_USART_HARDWARE_FLOW_CONTROL(CONTROL)`

4.1.6.5.7.1 Detailed Description

4.1.6.5.7.2 Macro Definition Documentation

IS_USART_HARDWARE_FLOW_CONTROL

```
#define IS_USART_HARDWARE_FLOW_CONTROL(  
    CONTROL )
```

Value:

```
((CONTROL) == USART_HardwareFlowControl_None) || \  
((CONTROL) == USART_HardwareFlowControl_RTS) || \  
((CONTROL) == USART_HardwareFlowControl_CTS) || \  
((CONTROL) == USART_HardwareFlowControl_RTS_CTS)
```

USART_HardwareFlowControl_CTS

```
#define USART_HardwareFlowControl_CTS ((uint16_t)0x0200)
```

USART_HardwareFlowControl_None

```
#define USART_HardwareFlowControl_None ((uint16_t)0x0000)
```

USART_HardwareFlowControl_RTS

```
#define USART_HardwareFlowControl_RTS ((uint16_t)0x0100)
```

USART_HardwareFlowControl_RTS_CTS

```
#define USART_HardwareFlowControl_RTS_CTS ((uint16_t)0x0300)
```

4.1.6.5.8 USART_Clock

Macros

- `#define USART_Clock_Disable ((uint16_t)0x0000)`
- `#define USART_Clock_Enable ((uint16_t)0x0800)`
- `#define IS_USART_CLOCK(CLOCK)`

4.1.6.5.8.1 Detailed Description**4.1.6.5.8.2 Macro Definition Documentation****IS_USART_CLOCK**

```
#define IS_USART_CLOCK(  
    CLOCK )
```

Value:

```
((CLOCK) == USART_Clock_Disable) || \  
((CLOCK) == USART_Clock_Enable))
```

USART_Clock_Disable

```
#define USART_Clock_Disable ((uint16_t)0x0000)
```

USART_Clock_Enable

```
#define USART_Clock_Enable ((uint16_t)0x0800)
```

4.1.6.5.9 USART_Clock_Polarity**Macros**

- #define [USART_CPOL_Low](#) ((uint16_t)0x0000)
- #define [USART_CPOL_High](#) ((uint16_t)0x0400)
- #define [IS_USART_CPOL](#)(CPOL) (((CPOL) == [USART_CPOL_Low](#)) || ((CPOL) == [USART_CPOL_High](#)))

4.1.6.5.9.1 Detailed Description**4.1.6.5.9.2 Macro Definition Documentation****IS_USART_CPOL**

```
#define IS_USART_CPOL(  
    CPOL ) (((CPOL) == USART\_CPOL\_Low) || ((CPOL) == USART\_CPOL\_High))
```

USART_CPOL_High

```
#define USART_CPOL_High ((uint16_t)0x0400)
```

USART_CPOL_Low

```
#define USART_CPOL_Low ((uint16_t)0x0000)
```

4.1.6.5.10 USART_Clock_Phase**Macros**

- #define [USART_CPHA_1Edge](#) ((uint16_t)0x0000)
- #define [USART_CPHA_2Edge](#) ((uint16_t)0x0200)
- #define [IS_USART_CPHA](#)(CPHA) (((CPHA) == [USART_CPHA_1Edge](#)) || ((CPHA) == [USART_CPHA_2Edge](#)))

4.1.6.5.10.1 Detailed Description**4.1.6.5.10.2 Macro Definition Documentation****IS_USART_CPHA**

```
#define IS_USART_CPHA(  
    CPHA ) (((CPHA) == USART\_CPHA\_1Edge) || ((CPHA) == USART\_CPHA\_2Edge))
```

USART_CPHA_1Edge

```
#define USART_CPHA_1Edge ((uint16_t)0x0000)
```

USART_CPHA_2Edge

```
#define USART_CPHA_2Edge ((uint16_t)0x0200)
```

4.1.6.5.11 USART_Last_Bit**Macros**

- `#define USART_LastBit_Disable ((uint16_t)0x0000)`
- `#define USART_LastBit_Enable ((uint16_t)0x0100)`
- `#define IS_USART_LASTBIT(LASTBIT)`

4.1.6.5.11.1 Detailed Description**4.1.6.5.11.2 Macro Definition Documentation****IS_USART_LASTBIT**

```
#define IS_USART_LASTBIT(  
    LASTBIT )
```

Value:

```
((LASTBIT) == USART_LastBit_Disable) || \  
(LASTBIT) == USART_LastBit_Enable)
```

USART_LastBit_Disable

```
#define USART_LastBit_Disable ((uint16_t)0x0000)
```

USART_LastBit_Enable

```
#define USART_LastBit_Enable ((uint16_t)0x0100)
```

4.1.6.5.12 USART_Interrupt_definition**Modules**

- [USART_Legacy](#)

Macros

- `#define USART_IT_PE ((uint16_t)0x0028)`
- `#define USART_IT_TXE ((uint16_t)0x0727)`
- `#define USART_IT_TC ((uint16_t)0x0626)`
- `#define USART_IT_RXNE ((uint16_t)0x0525)`
- `#define USART_IT_ORE_RX ((uint16_t)0x0325) /* In case interrupt is generated if the RXNEIE bit is set */`
- `#define USART_IT_IDLE ((uint16_t)0x0424)`
- `#define USART_IT_LBD ((uint16_t)0x0846)`
- `#define USART_IT_CTS ((uint16_t)0x096A)`
- `#define USART_IT_ERR ((uint16_t)0x0060)`
- `#define USART_IT_ORE_ER ((uint16_t)0x0360) /* In case interrupt is generated if the EIE bit is set */`
- `#define USART_IT_NE ((uint16_t)0x0260)`
- `#define USART_IT_FE ((uint16_t)0x0160)`
- `#define IS_USART_CONFIG_IT(IT)`
- `#define IS_USART_GET_IT(IT)`
- `#define IS_USART_CLEAR_IT(IT)`

4.1.6.5.12.1 Detailed Description**4.1.6.5.12.2 Macro Definition Documentation****IS_USART_CLEAR_IT**

```
#define IS_USART_CLEAR_IT(  
    IT )
```

Value:

```
((IT) == USART_IT_TC) || ((IT) == USART_IT_RXNE) || \  
((IT) == USART_IT_LBD) || ((IT) == USART_IT_CTS))
```

IS_USART_CONFIG_IT

```
#define IS_USART_CONFIG_IT(  
    IT )
```

Value:

```
((IT) == USART_IT_PE) || ((IT) == USART_IT_TXE) || \  
((IT) == USART_IT_TC) || ((IT) == USART_IT_RXNE) || \  
((IT) == USART_IT_IDLE) || ((IT) == USART_IT_LBD) || \  
((IT) == USART_IT_CTS) || ((IT) == USART_IT_ERR))
```

IS_USART_GET_IT

```
#define IS_USART_GET_IT(  
    IT )
```

Value:

```
((IT) == USART_IT_PE) || ((IT) == USART_IT_TXE) || \  
((IT) == USART_IT_TC) || ((IT) == USART_IT_RXNE) || \  
((IT) == USART_IT_IDLE) || ((IT) == USART_IT_LBD) || \  
((IT) == USART_IT_CTS) || ((IT) == USART_IT_ORE) || \  
((IT) == USART_IT_ORE_RX) || ((IT) == USART_IT_ORE_ER) || \  
((IT) == USART_IT_NE) || ((IT) == USART_IT_FE))
```

USART_IT_CTS

```
#define USART_IT_CTS ((uint16_t)0x096A)
```

USART_IT_ERR

```
#define USART_IT_ERR ((uint16_t)0x0060)
```

USART_IT_FE

```
#define USART_IT_FE ((uint16_t)0x0160)
```

USART_IT_IDLE

```
#define USART_IT_IDLE ((uint16_t)0x0424)
```

USART_IT_LBD

```
#define USART_IT_LBD ((uint16_t)0x0846)
```

USART_IT_NE

```
#define USART_IT_NE ((uint16_t)0x0260)
```

USART_IT_ORE_ER

```
#define USART_IT_ORE_ER ((uint16_t)0x0360) /* In case interrupt is generated if the EIE bit is  
set */
```

USART_IT_ORE_RX

```
#define USART_IT_ORE_RX ((uint16_t)0x0325) /* In case interrupt is generated if the RXNEIE bit
is set */
```

USART_IT_PE

```
#define USART_IT_PE ((uint16_t)0x0028)
```

USART_IT_RXNE

```
#define USART_IT_RXNE ((uint16_t)0x0525)
```

USART_IT_TC

```
#define USART_IT_TC ((uint16_t)0x0626)
```

USART_IT_TXE

```
#define USART_IT_TXE ((uint16_t)0x0727)
```

4.1.6.5.12.3 USART_Legacy**Macros**

- `#define USART_IT_ORE USART_IT_ORE_ER`

Detailed Description**Macro Definition Documentation****USART_IT_ORE**

```
#define USART_IT_ORE USART_IT_ORE_ER
```

4.1.6.5.13 USART_DMA_Requests**Macros**

- `#define USART_DMAREq_Tx ((uint16_t)0x0080)`
- `#define USART_DMAREq_Rx ((uint16_t)0x0040)`
- `#define IS_USART_DMAREQ(DMAREQ) (((DMAREQ) & (uint16_t)0xFF3F) == 0x00) && ((DMAREQ) != (uint16_t)0x00)`

4.1.6.5.13.1 Detailed Description**4.1.6.5.13.2 Macro Definition Documentation****IS_USART_DMAREQ**

```
#define IS_USART_DMAREQ(  
    DMAREQ ) (((DMAREQ) & (uint16_t)0xFF3F) == 0x00) && ((DMAREQ) != (uint16_t)0x00))
```

USART_DMAREq_Rx

```
#define USART_DMAREq_Rx ((uint16_t)0x0040)
```

USART_DMAREq_Tx

```
#define USART_DMAREq_Tx ((uint16_t)0x0080)
```

4.1.6.5.14 USART_WakeUp_methods

Macros

- #define [USART_WakeUp_IdleLine](#) ((uint16_t)0x0000)
- #define [USART_WakeUp_AddressMark](#) ((uint16_t)0x0800)
- #define [IS_USART_WAKEUP](#)(WAKEUP)

4.1.6.5.14.1 Detailed Description

4.1.6.5.14.2 Macro Definition Documentation

IS_USART_WAKEUP

```
#define IS_USART_WAKEUP(  
    WAKEUP )
```

Value:

```
((WAKEUP) == USART_WakeUp_IdleLine) || \  
((WAKEUP) == USART_WakeUp_AddressMark))
```

USART_WakeUp_AddressMark

```
#define USART_WakeUp_AddressMark ((uint16_t)0x0800)
```

USART_WakeUp_IdleLine

```
#define USART_WakeUp_IdleLine ((uint16_t)0x0000)
```

4.1.6.5.15 USART_LIN_Break_Detection_Length

Macros

- #define [USART_LINBreakDetectLength_10b](#) ((uint16_t)0x0000)
- #define [USART_LINBreakDetectLength_11b](#) ((uint16_t)0x0020)
- #define [IS_USART_LIN_BREAK_DETECT_LENGTH](#)(LENGTH)

4.1.6.5.15.1 Detailed Description

4.1.6.5.15.2 Macro Definition Documentation

IS_USART_LIN_BREAK_DETECT_LENGTH

```
#define IS_USART_LIN_BREAK_DETECT_LENGTH(  
    LENGTH )
```

Value:

```
((LENGTH) == USART_LINBreakDetectLength_10b) || \  
((LENGTH) == USART_LINBreakDetectLength_11b))
```

USART_LINBreakDetectLength_10b

```
#define USART_LINBreakDetectLength_10b ((uint16_t)0x0000)
```

USART_LINBreakDetectLength_11b

```
#define USART_LINBreakDetectLength_11b ((uint16_t)0x0020)
```

4.1.6.5.16 USART_IrDA_Low_Power

Macros

- #define [USART_IrDAMode_LowPower](#) ((uint16_t)0x0004)
- #define [USART_IrDAMode_Normal](#) ((uint16_t)0x0000)
- #define [IS_USART_IRDA_MODE](#)(MODE)

4.1.6.5.16.1 Detailed Description

4.1.6.5.16.2 Macro Definition Documentation

IS_USART_IRDA_MODE

```
#define IS_USART_IRDA_MODE(  
    MODE )
```

Value:

```
((MODE) == USART_IrDAMode_LowPower) || \  
((MODE) == USART_IrDAMode_Normal))
```

USART_IrDAMode_LowPower

```
#define USART_IrDAMode_LowPower ((uint16_t)0x0004)
```

USART_IrDAMode_Normal

```
#define USART_IrDAMode_Normal ((uint16_t)0x0000)
```

4.1.6.5.17 USART_Flags

Macros

- #define USART_FLAG_CTS ((uint16_t)0x0200)
- #define USART_FLAG_LBD ((uint16_t)0x0100)
- #define USART_FLAG_TXE ((uint16_t)0x0080)
- #define USART_FLAG_TC ((uint16_t)0x0040)
- #define USART_FLAG_RXNE ((uint16_t)0x0020)
- #define USART_FLAG_IDLE ((uint16_t)0x0010)
- #define USART_FLAG_ORE ((uint16_t)0x0008)
- #define USART_FLAG_NE ((uint16_t)0x0004)
- #define USART_FLAG_FE ((uint16_t)0x0002)
- #define USART_FLAG_PE ((uint16_t)0x0001)
- #define IS_USART_FLAG(*FLAG*)
- #define IS_USART_CLEAR_FLAG(*FLAG*) (((*FLAG*) & (uint16_t)0xFC9F) == 0x00) && ((*FLAG*) != (uint16_t)0x00)
- #define IS_USART_BAUDRATE(*BAUDRATE*) (((*BAUDRATE*) > 0) && ((*BAUDRATE*) < 7500001))
- #define IS_USART_ADDRESS(*ADDRESS*) ((*ADDRESS*) <= 0xF)
- #define IS_USART_DATA(*DATA*) ((*DATA*) <= 0xFF)

4.1.6.5.17.1 Detailed Description

4.1.6.5.17.2 Macro Definition Documentation

IS_USART_ADDRESS

```
#define IS_USART_ADDRESS(  
    ADDRESS ) ((ADDRESS) <= 0xF)
```

IS_USART_BAUDRATE

```
#define IS_USART_BAUDRATE(  
    BAUDRATE ) (((BAUDRATE) > 0) && ((BAUDRATE) < 7500001))
```

IS_USART_CLEAR_FLAG

```
#define IS_USART_CLEAR_FLAG(  
    FLAG ) (((FLAG) & (uint16_t)0xFC9F) == 0x00) && ((FLAG) != (uint16_t)0x00)
```


IS_USART_DATA

```
#define IS_USART_DATA(  
    DATA ) ((DATA) <= 0x1FF)
```

IS_USART_FLAG

```
#define IS_USART_FLAG(  
    FLAG )
```

Value:

```
((FLAG) == USART_FLAG_PE) || ((FLAG) == USART_FLAG_TXE) || \  
((FLAG) == USART_FLAG_TC) || ((FLAG) == USART_FLAG_RXNE) || \  
((FLAG) == USART_FLAG_IDLE) || ((FLAG) == USART_FLAG_LBD) || \  
((FLAG) == USART_FLAG_CTS) || ((FLAG) == USART_FLAG_ORE) || \  
((FLAG) == USART_FLAG_NE) || ((FLAG) == USART_FLAG_FE)
```

USART_FLAG_CTS

```
#define USART_FLAG_CTS ((uint16_t)0x0200)
```

USART_FLAG_FE

```
#define USART_FLAG_FE ((uint16_t)0x0002)
```

USART_FLAG_IDLE

```
#define USART_FLAG_IDLE ((uint16_t)0x0010)
```

USART_FLAG_LBD

```
#define USART_FLAG_LBD ((uint16_t)0x0100)
```

USART_FLAG_NE

```
#define USART_FLAG_NE ((uint16_t)0x0004)
```

USART_FLAG_ORE

```
#define USART_FLAG_ORE ((uint16_t)0x0008)
```

USART_FLAG_PE

```
#define USART_FLAG_PE ((uint16_t)0x0001)
```

USART_FLAG_RXNE

```
#define USART_FLAG_RXNE ((uint16_t)0x0020)
```

USART_FLAG_TC

```
#define USART_FLAG_TC ((uint16_t)0x0040)
```

USART_FLAG_TXE

```
#define USART_FLAG_TXE ((uint16_t)0x0080)
```

4.2 CMSIS

Modules

- [Stm32f4xx_system](#)

4.2.1 Detailed Description

4.2.2 Stm32f4xx_system

Modules

- [STM32F4xx_System_Private_Includes](#)
- [STM32F4xx_System_Private_TypesDefinitions](#)
- [STM32F4xx_System_Private_Defines](#)
- [STM32F4xx_System_Private_Macros](#)
- [STM32F4xx_System_Private_Variables](#)
- [STM32F4xx_System_Private_FunctionPrototypes](#)
- [STM32F4xx_System_Private_Functions](#)

4.2.2.1 Detailed Description

4.2.2.2 STM32F4xx_System_Private_Includes

Macros

- `#define HSE_VALUE ((uint32_t)25000000)`
- `#define HSI_VALUE ((uint32_t)16000000)`

4.2.2.2.1 Detailed Description

4.2.2.2.2 Macro Definition Documentation

4.2.2.2.2.1 HSE_VALUE

```
#define HSE_VALUE ((uint32_t)25000000)
```

Default value of the External oscillator in Hz

4.2.2.2.2.2 HSI_VALUE

```
#define HSI_VALUE ((uint32_t)16000000)
```

Value of the Internal oscillator in Hz

4.2.2.3 STM32F4xx_System_Private_TypesDefinitions

4.2.2.4 STM32F4xx_System_Private_Defines

4.2.2.5 STM32F4xx_System_Private_Macros

4.2.2.6 STM32F4xx_System_Private_Variables

Variables

- `uint32_t SystemCoreClock = 16000000`
- `const uint8_t AHBPrescTable[16] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}`
- `const uint8_t APBPrescTable[8] = {0, 0, 0, 0, 1, 2, 3, 4}`

4.2.2.6.1 Detailed Description

4.2.2.6.2 Variable Documentation

4.2.2.6.2.1 AHBPrescTable

```
const uint8_t AHBPrescTable[16] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
```

4.2.2.6.2.2 APBPrescTable

```
const uint8_t APBPrescTable[8] = {0, 0, 0, 0, 1, 2, 3, 4}
```

4.2.2.6.2.3 SystemCoreClock

```
uint32_t SystemCoreClock = 16000000
```

4.2.2.7 STM32F4xx_System_Private_FunctionPrototypes

4.2.2.8 STM32F4xx_System_Private_Functions

Functions

- void [SystemInit](#) (void)
Setup the microcontroller system Initialize the FPU setting, vector table location and External memory configuration.
- void [SystemCoreClockUpdate](#) (void)
Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

4.2.2.8.1 Detailed Description

4.2.2.8.2 Function Documentation

4.2.2.8.2.1 SystemCoreClockUpdate()

```
void SystemCoreClockUpdate (
    void )
```

Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Note

Each time the core clock (HCLK) changes, this function must be called to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:

- If SYSCLK source is HSI, SystemCoreClock will contain the [HSI_VALUE\(*\)](#)
- If SYSCLK source is HSE, SystemCoreClock will contain the [HSE_VALUE\(**\)](#)
- If SYSCLK source is PLL, SystemCoreClock will contain the [HSE_VALUE\(**\)](#) or [HSI_VALUE\(*\)](#) multiplied/divided by the PLL factors.

(*) HSI_VALUE is a constant defined in stm32f4xx_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 stm32f4xx_hal_conf.h file (its value depends on the application requirements), 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.

Parameters

None	
------	--

Return values

None	
------	--

4.2.2.8.2.2 SystemInit()

```
void SystemInit (  
    void )
```

Setup the microcontroller system Initialize the FPU setting, vector table location and External memory configuration.

Parameters

None	
------	--

Return values

None	
------	--

Chapter 5

Data Structure Documentation

5.1 `_ADC_InitTypeDef` Struct Reference

```
#include <adc.h>
```

Data Fields

- `uint32_t` [ClockPrescaler](#)
- `uint32_t` [Resolution](#)
- `uint32_t` [DataAlign](#)
- `uint32_t` [ScanConvMode](#)
- `uint32_t` [EOCSelection](#)
- `uint32_t` [ContinuousConvMode](#)
- `uint32_t` [DMAContinuousRequests](#)
- `uint32_t` [NbrOfConversion](#)
- `uint32_t` [DiscontinuousConvMode](#)
- `uint32_t` [NbrOfDiscConversion](#)
- `uint32_t` [ExternalTrigConv](#)
- `uint32_t` [ExternalTrigConvEdge](#)

5.1.1 Field Documentation

5.1.1.1 `ClockPrescaler`

```
uint32_t ClockPrescaler
```

Select the frequency of the clock to the ADC. The clock is common for all the ADCs. This parameter can be a value of `ADC_ClockPrescaler`

5.1.1.2 `ContinuousConvMode`

```
uint32_t ContinuousConvMode
```

Specifies whether the conversion is performed in Continuous or Single mode. This parameter can be set to `ENABLE` or `DISABLE`.

5.1.1.3 `DataAlign`

```
uint32_t DataAlign
```

Specifies whether the ADC data alignment is left or right.
This parameter can be a value of [ADC_data_align](#)

5.1.1.4 `DiscontinuousConvMode`

```
uint32_t DiscontinuousConvMode
```

Specifies whether the conversion is performed in Discontinuous or not for regular channels. This parameter can be set to `ENABLE` or `DISABLE`.

5.1.1.5 DMAContinuousRequests

`uint32_t DMAContinuousRequests`

Specifies whether the DMA requests is performed in Continuous or in Single mode. This parameter can be set to ENABLE or DISABLE.

5.1.1.6 EOCSelction

`uint32_t EOCSelction`

Specifies whether the EOC flag is set at the end of single channel conversion or at the end of all conversions. This parameter can be a value of ADC_EOCSelction Note: Impact on overrun when not using DMA: When EOCSelction is set to ADC_EOC_SINGLE_CONV, overrun detection is automatically enabled, in this case each conversion data must be read. To perform ADC conversions without having to read all conversion data, this parameter must be set to ADC_EOC_SEQ_CONV

5.1.1.7 ExternalTrigConv

`uint32_t ExternalTrigConv`

Selects the external event used to trigger the conversion start of regular group. If set to ADC_SOFTWARE_START, external triggers are disabled. This parameter can be a value of ADC_External_trigger_Source_Regular Note: This parameter can be modified only if there is no conversion is ongoing.

5.1.1.8 ExternalTrigConvEdge

`uint32_t ExternalTrigConvEdge`

Selects the external trigger edge of regular group. If trigger is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of ADC_External_trigger_edge_Regular Note: This parameter can be modified only if there is no conversion is ongoing.

5.1.1.9 NbrOfConversion

`uint32_t NbrOfConversion`

Specifies the number of ADC conversions that will be done using the sequencer for regular channel group. This parameter must be a number between Min_Data = 1 and Max_Data = 16.

5.1.1.10 NbrOfDiscConversion

`uint32_t NbrOfDiscConversion`

Specifies the number of ADC discontinuous conversions that will be done using the sequencer for regular channel group. This parameter must be a number between Min_Data = 1 and Max_Data = 8.

5.1.1.11 Resolution

`uint32_t Resolution`

Configures the ADC resolution dual mode. This parameter can be a value of ADC_Resolution

5.1.1.12 ScanConvMode

`uint32_t ScanConvMode`

Specifies whether the conversion is performed in Scan (multi channels) or Single (one channel) mode. This parameter can be set to ENABLE or DISABLE

The documentation for this struct was generated from the following file:

- [drivers/adc.h](#)

5.2 ADC_ChannelConfTypeDef Struct Reference

```
#include <adc.h>
```

Data Fields

- uint32_t [Channel](#)
- uint32_t [Rank](#)
- uint32_t [SamplingTime](#)
- uint32_t [Offset](#)

5.2.1 Field Documentation**5.2.1.1 Channel**

uint32_t Channel

The ADC channel to configure. This parameter can be a value of [ADC_channels](#)

5.2.1.2 Offset

uint32_t Offset

Reserved for future use, can be set to 0

5.2.1.3 Rank

uint32_t Rank

The rank in the regular group sequencer. This parameter must be a number between Min_Data = 1 and Max_Data = 16

5.2.1.4 SamplingTime

uint32_t SamplingTime

The sample time value to be set for the selected channel. This parameter can be a value of [ADC_sampling_times](#)

The documentation for this struct was generated from the following file:

- drivers/[adc.h](#)

5.3 ADC_CommonInitTypeDef Struct Reference

ADC Common Init structure definition

```
#include <stm32f4xx_adc.h>
```

Data Fields

- uint32_t [ADC_Mode](#)
- uint32_t [ADC_Prescaler](#)
- uint32_t [ADC_DMAAccessMode](#)
- uint32_t [ADC_TwoSamplingDelay](#)

5.3.1 Detailed Description

ADC Common Init structure definition

5.3.2 Field Documentation**5.3.2.1 ADC_DMAAccessMode**

uint32_t ADC_DMAAccessMode

Configures the Direct memory access mode for multi ADC mode. This parameter can be a value of [ADC_Direct_memory_access_mode_for_multi_mode](#)

5.3.2.2 ADC_Mode

uint32_t ADC_Mode

Configures the ADC to operate in independent or multi mode. This parameter can be a value of [ADC_Common_mode](#)

5.3.2.3 ADC_Prescaler

uint32_t ADC_Prescaler

Select the frequency of the clock to the ADC. The clock is common for all the ADCs. This parameter can be a value of [ADC_Prescaler](#)

5.3.2.4 ADC_TwoSamplingDelay

uint32_t ADC_TwoSamplingDelay

Configures the Delay between 2 sampling phases. This parameter can be a value of [ADC_delay_between_2_sampling_phases](#)
The documentation for this struct was generated from the following file:

- [drivers/stm32f4xx_adc.h](#)

5.4 ADC_HandleTypeDef Struct Reference

```
#include <adc.h>
```

Data Fields

- ADC_TypeDef * [Instance](#)
- [_ADC_InitTypeDef](#) Init
- __IO uint32_t [NbrOfCurrentConversionRank](#)
- [HAL_LockTypeDef](#) Lock
- __IO [HAL_ADC_StateTypeDef](#) State
- __IO uint32_t [ErrorCode](#)

5.4.1 Field Documentation

5.4.1.1 ErrorCode

__IO uint32_t ErrorCode

ADC Error code

5.4.1.2 Init

[_ADC_InitTypeDef](#) Init

ADC required parameters

5.4.1.3 Instance

ADC_TypeDef* Instance

Register base address

5.4.1.4 Lock

[HAL_LockTypeDef](#) Lock

ADC locking object

5.4.1.5 NbrOfCurrentConversionRank

__IO uint32_t NbrOfCurrentConversionRank

ADC number of current conversion rank

5.4.1.6 State

__IO [HAL_ADC_StateTypeDef](#) State

ADC communication state

The documentation for this struct was generated from the following file:

- [drivers/adc.h](#)

5.5 ADC_InitTypeDef Struct Reference

ADC Init structure definition

```
#include <stm32f4xx_adc.h>
```

Data Fields

- uint32_t [ADC_Resolution](#)
- FunctionalState [ADC_ScanConvMode](#)
- FunctionalState [ADC_ContinuousConvMode](#)
- uint32_t [ADC_ExternalTrigConvEdge](#)
- uint32_t [ADC_ExternalTrigConv](#)
- uint32_t [ADC_DataAlign](#)
- uint8_t [ADC_NbrOfConversion](#)

5.5.1 Detailed Description

ADC Init structure definition

5.5.2 Field Documentation

5.5.2.1 ADC_ContinuousConvMode

FunctionalState [ADC_ContinuousConvMode](#)

Specifies whether the conversion is performed in Continuous or Single mode. This parameter can be set to ENABLE or DISABLE.

5.5.2.2 ADC_DataAlign

uint32_t [ADC_DataAlign](#)

Specifies whether the ADC data alignment is left or right. This parameter can be a value of [ADC_data_align](#)

5.5.2.3 ADC_ExternalTrigConv

uint32_t [ADC_ExternalTrigConv](#)

Select the external event used to trigger the start of conversion of a regular group. This parameter can be a value of [ADC_extrenal_trigger_sources_for_regular_channels_conversion](#)

5.5.2.4 ADC_ExternalTrigConvEdge

uint32_t [ADC_ExternalTrigConvEdge](#)

Select the external trigger edge and enable the trigger of a regular group. This parameter can be a value of [ADC_external_trigger_edge_for_regular_channels_conversion](#)

5.5.2.5 ADC_NbrOfConversion

uint8_t [ADC_NbrOfConversion](#)

Specifies the number of ADC conversions that will be done using the sequencer for regular channel group. This parameter must range from 1 to 16.

5.5.2.6 ADC_Resolution

`uint32_t ADC_Resolution`

Configures the ADC resolution dual mode. This parameter can be a value of [ADC_resolution](#)

5.5.2.7 ADC_ScanConvMode

`FunctionalState ADC_ScanConvMode`

Specifies whether the conversion is performed in Scan (multichannels) or Single (one channel) mode. This parameter can be set to ENABLE or DISABLE

The documentation for this struct was generated from the following file:

- [drivers/stm32f4xx_adc.h](#)

5.6 analogin_s Struct Reference

```
#include <adc.h>
```

Data Fields

- [ADCName](#) `adc`
- [Pin](#) `pin`
- `uint8_t` [channel](#)

5.6.1 Field Documentation

5.6.1.1 adc

[ADCName](#) `adc`

5.6.1.2 channel

`uint8_t` `channel`

5.6.1.3 pin

[Pin](#) `pin`

The documentation for this struct was generated from the following file:

- [drivers/adc.h](#)

5.7 GPIO_InitTypeDef Struct Reference

GPIO Init structure definition

```
#include <adc.h>
```

Data Fields

- `uint32_t` [Pin](#)
- `uint32_t` [Mode](#)
- `uint32_t` [Pull](#)
- `uint32_t` [Speed](#)
- `uint32_t` [Alternate](#)
- `uint32_t` [GPIO_Pin](#)
- [GPIOMode_TypeDef](#) [GPIO_Mode](#)
- [GPISpeed_TypeDef](#) [GPIO_Speed](#)
- [GPIOOType_TypeDef](#) [GPIO_OType](#)
- [GIOPuPd_TypeDef](#) [GPIO_PuPd](#)

5.7.1 Detailed Description

GPIO Init structure definition

5.7.2 Field Documentation

5.7.2.1 Alternate

`uint32_t Alternate`

Peripheral to be connected to the selected pins. This parameter can be a value of [GPIO_Alternate_function_selection](#)

5.7.2.2 GPIO_Mode

[GPIO_Mode_TypeDef](#) `GPIO_Mode`

Specifies the operating mode for the selected pins. This parameter can be a value of [GPIO_Mode_TypeDef](#)

5.7.2.3 GPIO_OType

[GPIO_OType_TypeDef](#) `GPIO_OType`

Specifies the operating output type for the selected pins. This parameter can be a value of [GPIO_OType_TypeDef](#)

5.7.2.4 GPIO_Pin

`uint32_t GPIO_Pin`

Specifies the GPIO pins to be configured. This parameter can be any value of [GPIO_pins_define](#)

5.7.2.5 GPIO_PuPd

[GPIO_PuPd_TypeDef](#) `GPIO_PuPd`

Specifies the operating Pull-up/Pull down for the selected pins. This parameter can be a value of [GPIO_PuPd_TypeDef](#)

5.7.2.6 GPIO_Speed

[GPIO_Speed_TypeDef](#) `GPIO_Speed`

Specifies the speed for the selected pins. This parameter can be a value of [GPIO_Speed_TypeDef](#)

5.7.2.7 Mode

`uint32_t Mode`

Specifies the operating mode for the selected pins. This parameter can be a value of [GPIO_mode_define](#)

5.7.2.8 Pin

`uint32_t Pin`

Specifies the GPIO pins to be configured. This parameter can be any value of [GPIO_pins_define](#)

5.7.2.9 Pull

`uint32_t Pull`

Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of [GPIO_pull_define](#)

5.7.2.10 Speed

`uint32_t Speed`

Specifies the speed for the selected pins. This parameter can be a value of [GPIO_speed_define](#)

The documentation for this struct was generated from the following files:

- [drivers/adc.h](#)
- [drivers/stm32f4xx_gpio.h](#)

5.8 I2C_InitTypeDef Struct Reference

I2C Init structure definition

```
#include <stm32f4xx_i2c.h>
```

Data Fields

- `uint32_t` [I2C_ClockSpeed](#)
- `uint16_t` [I2C_Mode](#)
- `uint16_t` [I2C_DutyCycle](#)
- `uint16_t` [I2C_OwnAddress1](#)
- `uint16_t` [I2C_Ack](#)
- `uint16_t` [I2C_AcknowledgedAddress](#)

5.8.1 Detailed Description

I2C Init structure definition

5.8.2 Field Documentation

5.8.2.1 I2C_Ack

`uint16_t` [I2C_Ack](#)

Enables or disables the acknowledgement. This parameter can be a value of [I2C_acknowledgement](#)

5.8.2.2 I2C_AcknowledgedAddress

`uint16_t` [I2C_AcknowledgedAddress](#)

Specifies if 7-bit or 10-bit address is acknowledged. This parameter can be a value of [I2C_acknowledged_address](#)

5.8.2.3 I2C_ClockSpeed

`uint32_t` [I2C_ClockSpeed](#)

Specifies the clock frequency. This parameter must be set to a value lower than 400kHz

5.8.2.4 I2C_DutyCycle

`uint16_t` [I2C_DutyCycle](#)

Specifies the I2C fast mode duty cycle. This parameter can be a value of [I2C_duty_cycle_in_fast_mode](#)

5.8.2.5 I2C_Mode

`uint16_t` [I2C_Mode](#)

Specifies the I2C mode. This parameter can be a value of [I2C_mode](#)

5.8.2.6 I2C_OwnAddress1

`uint16_t` [I2C_OwnAddress1](#)

Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.

The documentation for this struct was generated from the following file:

- [drivers/stm32f4xx_i2c.h](#)

5.9 PinMap Struct Reference

```
#include <adc.h>
```

Data Fields

- [Pin pin](#)
- [int peripheral](#)
- [int function](#)

5.9.1 Field Documentation

5.9.1.1 function

```
int function
```

5.9.1.2 peripheral

```
int peripheral
```

5.9.1.3 pin

```
Pin pin
```

The documentation for this struct was generated from the following file:

- [drivers/adc.h](#)

5.10 RCC_ClocksTypeDef Struct Reference

```
#include <stm32f4xx_rcc.h>
```

Data Fields

- [uint32_t SYSCLK_Frequency](#)
- [uint32_t HCLK_Frequency](#)
- [uint32_t PCLK1_Frequency](#)
- [uint32_t PCLK2_Frequency](#)

5.10.1 Field Documentation

5.10.1.1 HCLK_Frequency

```
uint32_t HCLK_Frequency
```

HCLK clock frequency expressed in Hz

5.10.1.2 PCLK1_Frequency

```
uint32_t PCLK1_Frequency
```

PCLK1 clock frequency expressed in Hz

5.10.1.3 PCLK2_Frequency

```
uint32_t PCLK2_Frequency
```

PCLK2 clock frequency expressed in Hz

5.10.1.4 SYSCLK_Frequency

uint32_t SYSCLK_Frequency

SYSCLK clock frequency expressed in Hz

The documentation for this struct was generated from the following file:

- [drivers/stm32f4xx_rcc.h](#)

5.11 USART_ClockInitTypeDef Struct Reference

USART Clock Init Structure definition

```
#include <stm32f4xx_usart.h>
```

Data Fields

- uint16_t [USART_Clock](#)
- uint16_t [USART_CPOL](#)
- uint16_t [USART_CPHA](#)
- uint16_t [USART_LastBit](#)

5.11.1 Detailed Description

USART Clock Init Structure definition

5.11.2 Field Documentation

5.11.2.1 USART_Clock

uint16_t USART_Clock

Specifies whether the USART clock is enabled or disabled. This parameter can be a value of [USART_Clock](#)

5.11.2.2 USART_CPHA

uint16_t USART_CPHA

Specifies the clock transition on which the bit capture is made. This parameter can be a value of [USART_Clock_Phase](#)

5.11.2.3 USART_CPOL

uint16_t USART_CPOL

Specifies the steady state of the serial clock. This parameter can be a value of [USART_Clock_Polarity](#)

5.11.2.4 USART_LastBit

uint16_t USART_LastBit

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](#)

The documentation for this struct was generated from the following file:

- [drivers/stm32f4xx_usart.h](#)

5.12 USART_InitTypeDef Struct Reference

USART Init Structure definition

```
#include <stm32f4xx_usart.h>
```

Data Fields

- uint32_t [USART_BaudRate](#)
- uint16_t [USART_WordLength](#)
- uint16_t [USART_StopBits](#)
- uint16_t [USART_Parity](#)
- uint16_t [USART_Mode](#)
- uint16_t [USART_HardwareFlowControl](#)

5.12.1 Detailed Description

USART Init Structure definition

5.12.2 Field Documentation**5.12.2.1 USART_BaudRate**

```
uint32_t USART_BaudRate
```

This member configures the USART communication baud rate. The baud rate is computed using the following formula:

- $\text{IntegerDivider} = ((\text{PCLKx}) / (8 * (\text{OVR8} + 1) * (\text{USART_InitStruct->USART_BaudRate})))$
- $\text{FractionalDivider} = ((\text{IntegerDivider} - ((\text{u32}) \text{IntegerDivider})) * 8 * (\text{OVR8} + 1)) + 0.5$ Where OVR8 is the "over-sampling by 8 mode" configuration bit in the CR1 register.

5.12.2.2 USART_HardwareFlowControl

```
uint16_t USART_HardwareFlowControl
```

Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of [USART_Hardware_Flow_Control](#)

5.12.2.3 USART_Mode

```
uint16_t USART_Mode
```

Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [USART_Mode](#)

5.12.2.4 USART_Parity

```
uint16_t USART_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).

5.12.2.5 USART_StopBits

```
uint16_t USART_StopBits
```

Specifies the number of stop bits transmitted. This parameter can be a value of [USART_Stop_Bits](#)

5.12.2.6 USART_WordLength

```
uint16_t USART_WordLength
```

Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [USART_Word_Length](#)

The documentation for this struct was generated from the following file:

- [drivers/stm32f4xx_usart.h](#)

Chapter 6

File Documentation

6.1 drivers/adc.c File Reference

```
#include <platform.h>
#include <stdlib.h>
#include <adc.h>
```

Functions

- void [adc_init](#) (Pin pin)
- uint32_t [pinmap_find_peripheral](#) (Pin pin)
- uint32_t [pinmap_peripheral](#) (Pin pin)
- void [analogin_init](#) (analogin_s *obj, Pin pin)
- void [_ADC_Init](#) (ADC_HandleTypeDef *hadc)
- uint32_t [pinmap_find_function](#) (Pin pin)
- uint32_t [pinmap_function](#) (Pin pin)
 - Initializes the analogue to digital converter, and configures the appropriate GPIO pin.*
- void [pinmap_pinout](#) (Pin pin)
- void [pin_function](#) (Pin pin, int data)
- void [_GPIO_Init](#) (GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init)
- uint16_t [adc_read](#) (Pin pin)
- uint16_t [_adc_read](#) (analogin_s *obj)
 - Reads the current value of the ADC.*
- uint32_t [_ADC_GetValue](#) (ADC_HandleTypeDef *hadc)
- int [_ADC_PollForConversion](#) (ADC_HandleTypeDef *hadc, uint32_t Timeout)
- void [_ADC_ConfigChannel](#) (ADC_HandleTypeDef *hadc, ADC_ChannelConfTypeDef *sConfig)
- void [_ADC_Start](#) (ADC_HandleTypeDef *hadc)

Variables

- [ADC_HandleTypeDef](#) AdcHandle
- const [PinMap](#) PinMap_ADC []

6.1.1 Function Documentation

6.1.1.1 _ADC_ConfigChannel()

```
void _ADC_ConfigChannel (
    ADC_HandleTypeDef * hadc,
    ADC_ChannelConfTypeDef * sConfig )
```

6.1.1.2 `_ADC_GetValue()`

```
uint32_t _ADC_GetValue (
    ADC_HandleTypeDef * hadc )
```

6.1.1.3 `_ADC_Init()`

```
void _ADC_Init (
    ADC_HandleTypeDef * hadc )
```

6.1.1.4 `_ADC_PollForConversion()`

```
int _ADC_PollForConversion (
    ADC_HandleTypeDef * hadc,
    uint32_t Timeout )
```

6.1.1.5 `_adc_read()`

```
uint16_t _adc_read (
    analogin_s * obj )
```

Reads the current value of the ADC.

Returns

Potential of the pin, relative to ground.

6.1.1.6 `_ADC_Start()`

```
void _ADC_Start (
    ADC_HandleTypeDef * hadc )
```

6.1.1.7 `_GPIO_Init()`

```
void _GPIO_Init (
    GPIO_TypeDef * GPIOx,
    GPIO_InitTypeDef * GPIO_Init )
```

6.1.1.8 `adc_init()`

```
void adc_init (
    Pin pin )
```

6.1.1.9 `adc_read()`

```
uint16_t adc_read (
    Pin pin )
```

6.1.1.10 `analogin_init()`

```
void analogin_init (
    analogin_s * obj,
    Pin pin )
```

6.1.1.11 `pin_function()`

```
void pin_function (
    Pin pin,
    int data )
```

6.1.1.12 pinmap_find_function()

```
uint32_t pinmap_find_function (
    Pin pin )
```

6.1.1.13 pinmap_find_peripheral()

```
uint32_t pinmap_find_peripheral (
    Pin pin )
```

6.1.1.14 pinmap_function()

```
uint32_t pinmap_function (
    Pin pin )
```

Initializes the analogue to digital converter, and configures the appropriate GPIO pin.

6.1.1.15 pinmap_peripheral()

```
uint32_t pinmap_peripheral (
    Pin pin )
```

6.1.1.16 pinmap_pinout()

```
void pinmap_pinout (
    Pin pin )
```

6.1.2 Variable Documentation**6.1.2.1 AdcHandle**

`ADC_HandleTypeDef` AdcHandle

6.1.2.2 PinMap_ADC

```
const PinMap PinMap_ADC[]
```

Initial value:

```
= {
    {PA_0, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 0, 0)},
    {PA_1, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 1, 0)},
    {PA_2, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 2, 0)},
    {PA_3, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 3, 0)},
    {PA_4, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 4, 0)},
    {PA_5, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 5, 0)},
    {PA_6, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 6, 0)},
    {PA_7, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 7, 0)},
    {PB_0, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 8, 0)},
    {PB_1, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 9, 0)},
    {PC_0, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 10, 0)},
    {PC_1, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 11, 0)},
    {PC_2, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 12, 0)},
    {PC_3, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 13, 0)},
    {PC_4, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 14, 0)},
    {PC_5, (int)ADC1_BASE, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 15, 0)},
    {NC, 0, 0}
}
```

6.2 drivers/adc.h File Reference

Internal analogue to digital converter (ADC) controller.

```
#include "stm32f4xx_adc.h"
```

Data Structures

- struct `_ADC_InitTypeDef`

- struct [ADC_HandleTypeDef](#)
- struct [GPIO_InitTypeDef](#)

GPIO Init structure definition

- struct [PinMap](#)
- struct [analogin_s](#)
- struct [ADC_ChannelConfTypeDef](#)

Macros

- #define [ADC1_BASE](#) (APB2PERIPH_BASE + 0x2000)
- #define [STM_PIN_DATA_EXT](#)(MODE, PUPD, AFNUM, CHANNEL, INVERTED) (((int)(((INVERTED & 0x01) << 15) | ((CHANNEL & 0x0F) << 11) | ((AFNUM & 0x0F) << 7) | ((PUPD & 0x07) << 4) | ((MODE & 0x0F) << 0)))
- #define [STM_MODE_ANALOG](#) (5)
- #define [STM_PIN_CHANNEL](#)(X) (((X) >> 11) & 0x0F)
- #define [STM_PIN_MODE](#)(X) (((X) >> 0) & 0x0F)
- #define [STM_PIN_PUPD](#)(X) (((X) >> 4) & 0x07)
- #define [STM_PIN_AFNUM](#)(X) (((X) >> 7) & 0x0F)
- #define [STM_PORT](#)(X) (((uint32_t)(X) >> 4) & 0xF)
- #define [STM_PIN](#)(X) ((uint32_t)(X) & 0xF)
- #define [GPIO_NOPULL](#) ((uint32_t)0x00000000)
- #define [GPIO_PULLUP](#) ((uint32_t)0x00000001)
- #define [GPIO_PULLDOWN](#) ((uint32_t)0x00000002)
- #define [UNUSED](#)(x) ((void)(x))
- #define [GPIOA_BASE](#) (AHB1PERIPH_BASE + 0x0000)
- #define [GPIO_SPEED_LOW](#) ((uint32_t)0x00000000)
- #define [GPIO_SPEED_MEDIUM](#) ((uint32_t)0x00000001)
- #define [GPIO_SPEED_FAST](#) ((uint32_t)0x00000002)
- #define [GPIO_SPEED_HIGH](#) ((uint32_t)0x00000003)
- #define [GPIO_MODE](#) ((uint32_t)0x00000003)
- #define [ADC_CR1_SCANCONV](#)(_SCANCONV_MODE_) ((_SCANCONV_MODE_) << 8)
- #define [ADC_SOFTWARE_START](#) ((uint32_t)ADC_CR2_EXTSEL + 1)
- #define [ADC_CR2_CONTINUOUS](#)(_CONTINUOUS_MODE_) ((_CONTINUOUS_MODE_) << 1)
- #define [ADC_CR1_DISCONTINUOUS](#)(_NBR_DISCONTINUOUSCONV_) (((_NBR_DISCONTINUOUSCONV_ << POSITION_VAL(ADC_CR1_DISCNUM)) - 1) << POSITION_VAL(ADC_CR1_DISCNUM))
- #define [ADC_SQR1_NbrOfConversion](#)(_NbrOfConversion_) (((_NbrOfConversion_) - (uint8_t)1) << 20)
- #define [ADC_CR2_DMAContReq](#)(_DMAContReq_MODE_) ((_DMAContReq_MODE_) << 9)
- #define [ADC_CR2_EOCSelection](#)(_EOCSelection_MODE_) ((_EOCSelection_MODE_) << 10)
- #define [ADC_SMPR1](#)(_SAMPLETIME_, _CHANNELNB_) ((_SAMPLETIME_) << (3 * (((uint32_t)((uint16_t)_CHANNELNB_)) - 10)))
- #define [ADC_SMPR2](#)(_SAMPLETIME_, _CHANNELNB_) ((_SAMPLETIME_) << (3 * (((uint32_t)((uint16_t)_CHANNELNB_)) - 10)))
- #define [ADC_SQR3_RK](#)(_CHANNELNB_, _RANKNB_) (((uint32_t)((uint16_t)_CHANNELNB_)) << (5 * ((_RANKNB_) - 1)))
- #define [ADC_SQR2_RK](#)(_CHANNELNB_, _RANKNB_) (((uint32_t)((uint16_t)_CHANNELNB_)) << (5 * ((_RANKNB_) - 7)))
- #define [ADC_SQR1_RK](#)(_CHANNELNB_, _RANKNB_) (((uint32_t)((uint16_t)_CHANNELNB_)) << (5 * ((_RANKNB_) - 13)))
- #define [ADC_CHANNEL_VBAT](#) ((uint32_t)ADC_CHANNEL_18)
- #define [ADC_CHANNEL_VREFINT](#) ((uint32_t)ADC_CHANNEL_17)
- #define [ADC_CHANNEL_TEMPSENSOR](#) ((uint32_t)ADC_CHANNEL_16)
- #define [ADC_STAB_DELAY_US](#) ((uint32_t)3)
- #define [_IS_BIT_CLR](#)(REG, BIT) (((REG) & (BIT)) == RESET)
- #define [_IS_BIT_SET](#)(REG, BIT) (((REG) & (BIT)) != RESET)

- `#define _ADC_GET_FLAG(__HANDLE__, __FLAG__) (((__HANDLE__)->Instance->SR) & (__FLAG__))`
`== (__FLAG__)`
- `#define RCC_GPIOA_CLK_ENABLE()`
- `#define RCC_GPIOB_CLK_ENABLE()`
- `#define RCC_GPIOC_CLK_ENABLE()`
- `#define RCC_ADC1_CLK_ENABLE()`
- `#define ADC_CHANNEL_0 ((uint32_t)0x00000000)`
- `#define ADC_CHANNEL_1 ((uint32_t)ADC_CR1_AWDCH_0)`
- `#define ADC_CHANNEL_2 ((uint32_t)ADC_CR1_AWDCH_1)`
- `#define ADC_CHANNEL_3 ((uint32_t)(ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))`
- `#define ADC_CHANNEL_4 ((uint32_t)ADC_CR1_AWDCH_2)`
- `#define ADC_CHANNEL_5 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_0))`
- `#define ADC_CHANNEL_6 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1))`
- `#define ADC_CHANNEL_7 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))`
- `#define ADC_CHANNEL_8 ((uint32_t)ADC_CR1_AWDCH_3)`
- `#define ADC_CHANNEL_9 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_0))`
- `#define ADC_CHANNEL_10 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_1))`
- `#define ADC_CHANNEL_11 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))`
- `#define ADC_CHANNEL_12 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2))`
- `#define ADC_CHANNEL_13 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_0))`
- `#define ADC_CHANNEL_14 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1))`
- `#define ADC_CHANNEL_15 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))`
- `#define ADC_CHANNEL_16 ((uint32_t)ADC_CR1_AWDCH_4)`
- `#define ADC_CHANNEL_17 ((uint32_t)(ADC_CR1_AWDCH_4 | ADC_CR1_AWDCH_0))`
- `#define ADC_CHANNEL_18 ((uint32_t)(ADC_CR1_AWDCH_4 | ADC_CR1_AWDCH_1))`

Enumerations

- enum `HAL_LockTypeDef` { `HAL_UNLOCKED` = 0x00 , `HAL_LOCKED` = 0x01 }
- enum `HAL_ADC_StateTypeDef` {
`HAL_ADC_STATE_RESET` = 0x00 , `HAL_ADC_STATE_READY` = 0x01 , `HAL_ADC_STATE_BUSY` = 0x02 ,
`HAL_ADC_STATE_BUSY_REG` = 0x12 ,
`HAL_ADC_STATE_BUSY_INJ` = 0x22 , `HAL_ADC_STATE_BUSY_INJ_REG` = 0x32 , `HAL_ADC_STATE_TIMEOUT`
= 0x03 , `HAL_ADC_STATE_ERROR` = 0x04 ,
`HAL_ADC_STATE_EOC` = 0x05 , `HAL_ADC_STATE_EOC_REG` = 0x15 , `HAL_ADC_STATE_EOC_INJ` =
0x25 , `HAL_ADC_STATE_EOC_INJ_REG` = 0x35 ,
`HAL_ADC_STATE_AWD` = 0x06 }
- enum `PortName` {
`PortA` = 0 , `PortB` = 1 , `PortC` = 2 , `PortD` = 3 ,
`PortE` = 4 , `PortH` = 7 }
- enum `ADCName` { `ADC_1` = (int)ADC1_BASE }

Functions

- `uint32_t pinmap_function (Pin pin)`
Initializes the analogue to digital converter, and configures the appropriate GPIO pin.
- `void analogin_init (analogin_s *obj, Pin pin)`
- `uint32_t pinmap_peripheral (Pin pin)`
- `uint32_t pinmap_find_peripheral (Pin pin)`
- `void adc_init (Pin pin)`
- `void pinmap_pinout (Pin pin)`

- void [pin_function](#) ([Pin](#) pin, int data)
- void [_GPIO_Init](#) ([GPIO_TypeDef](#) *GPIOx, [GPIO_InitTypeDef](#) *GPIO_Init)
- void [_ADC_Init](#) ([ADC_HandleTypeDef](#) *hadc)
- void [_ADC_ConfigChannel](#) ([ADC_HandleTypeDef](#) *hadc, [ADC_ChannelConfTypeDef](#) *sConfig)
- void [_ADC_Start](#) ([ADC_HandleTypeDef](#) *hadc)
- int [_ADC_PollForConversion](#) ([ADC_HandleTypeDef](#) *hadc, uint32_t Timeout)
- uint32_t [_ADC_GetValue](#) ([ADC_HandleTypeDef](#) *hadc)
- uint16_t [_adc_read](#) ([analogin_s](#) *obj)
Reads the current value of the ADC.
- uint16_t [adc_read](#) ([Pin](#) pin)

6.2.1 Detailed Description

Internal analogue to digital converter (ADC) controller.

Copyright

ARM University Program © ARM Ltd 2014.

6.2.2 Macro Definition Documentation

6.2.2.1 _ADC_GET_FLAG

```
#define _ADC_GET_FLAG(  
    __HANDLE__,  
    __FLAG__ ) (((__HANDLE__)->Instance->SR) & (__FLAG__)) == (__FLAG__)
```

6.2.2.2 _IS_BIT_CLR

```
#define _IS_BIT_CLR(  
    REG,  
    BIT ) (((REG) & (BIT)) == RESET)
```

6.2.2.3 _IS_BIT_SET

```
#define _IS_BIT_SET(  
    REG,  
    BIT ) (((REG) & (BIT)) != RESET)
```

6.2.2.4 ADC1_BASE

```
#define ADC1_BASE (APB2PERIPH_BASE + 0x2000)
```

6.2.2.5 ADC_CHANNEL_0

```
#define ADC_CHANNEL_0 ((uint32_t)0x00000000)
```

6.2.2.6 ADC_CHANNEL_1

```
#define ADC_CHANNEL_1 ((uint32_t)ADC_CR1_AWDCH_0)
```

6.2.2.7 ADC_CHANNEL_10

```
#define ADC_CHANNEL_10 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_1))
```

6.2.2.8 ADC_CHANNEL_11

```
#define ADC_CHANNEL_11 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))
```

6.2.2.9 ADC_CHANNEL_12

```
#define ADC_CHANNEL_12 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2))
```

6.2.2.10 ADC_CHANNEL_13

```
#define ADC_CHANNEL_13 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_0))
```

6.2.2.11 ADC_CHANNEL_14

```
#define ADC_CHANNEL_14 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1))
```

6.2.2.12 ADC_CHANNEL_15

```
#define ADC_CHANNEL_15 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))
```

6.2.2.13 ADC_CHANNEL_16

```
#define ADC_CHANNEL_16 ((uint32_t)ADC_CR1_AWDCH_4)
```

6.2.2.14 ADC_CHANNEL_17

```
#define ADC_CHANNEL_17 ((uint32_t)(ADC_CR1_AWDCH_4 | ADC_CR1_AWDCH_0))
```

6.2.2.15 ADC_CHANNEL_18

```
#define ADC_CHANNEL_18 ((uint32_t)(ADC_CR1_AWDCH_4 | ADC_CR1_AWDCH_1))
```

6.2.2.16 ADC_CHANNEL_2

```
#define ADC_CHANNEL_2 ((uint32_t)ADC_CR1_AWDCH_1)
```

6.2.2.17 ADC_CHANNEL_3

```
#define ADC_CHANNEL_3 ((uint32_t)(ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))
```

6.2.2.18 ADC_CHANNEL_4

```
#define ADC_CHANNEL_4 ((uint32_t)ADC_CR1_AWDCH_2)
```

6.2.2.19 ADC_CHANNEL_5

```
#define ADC_CHANNEL_5 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_0))
```

6.2.2.20 ADC_CHANNEL_6

```
#define ADC_CHANNEL_6 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1))
```

6.2.2.21 ADC_CHANNEL_7

```
#define ADC_CHANNEL_7 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))
```

6.2.2.22 ADC_CHANNEL_8

```
#define ADC_CHANNEL_8 ((uint32_t)ADC_CR1_AWDCH_3)
```

6.2.2.23 ADC_CHANNEL_9

```
#define ADC_CHANNEL_9 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_0))
```

6.2.2.24 ADC_CHANNEL_TEMPSENSOR

```
#define ADC_CHANNEL_TEMPSENSOR ((uint32_t)ADC_CHANNEL_16)
```

6.2.2.25 ADC_CHANNEL_VBAT

```
#define ADC_CHANNEL_VBAT ((uint32_t)ADC_CHANNEL_18)
```

6.2.2.26 ADC_CHANNEL_VREFINT

```
#define ADC_CHANNEL_VREFINT ((uint32_t)ADC_CHANNEL_17)
```

6.2.2.27 ADC_CR1_DISCONTINUOUS

```
#define ADC_CR1_DISCONTINUOUS(
    _NBR_DISCONTINUOUSCONV_ ) (((_NBR_DISCONTINUOUSCONV_) - 1) << POSITION_VAL(ADC_↵
    _CR1_DISCNUM))
```

6.2.2.28 ADC_CR1_SCANCONV

```
#define ADC_CR1_SCANCONV(
    _SCANCONV_MODE_ ) ((_SCANCONV_MODE_) << 8)
```

6.2.2.29 ADC_CR2_CONTINUOUS

```
#define ADC_CR2_CONTINUOUS(
    _CONTINUOUS_MODE_ ) ((_CONTINUOUS_MODE_) << 1)
```

6.2.2.30 ADC_CR2_DMAContReq

```
#define ADC_CR2_DMAContReq(
    _DMAContReq_MODE_ ) ((_DMAContReq_MODE_) << 9)
```

6.2.2.31 ADC_CR2_EOCSelection

```
#define ADC_CR2_EOCSelection(
    _EOCSelection_MODE_ ) ((_EOCSelection_MODE_) << 10)
```

6.2.2.32 ADC_SMPR1

```
#define ADC_SMPR1(
    _SAMPLETIME_,
    _CHANNELNB_ ) ((_SAMPLETIME_) << (3 * (((uint32_t)((uint16_t)(_CHANNELNB_))) -
    10)))
```

6.2.2.33 ADC_SMPR2

```
#define ADC_SMPR2(
    _SAMPLETIME_,
    _CHANNELNB_ ) ((_SAMPLETIME_) << (3 * (((uint32_t)((uint16_t)(_CHANNELNB_))))))
```

6.2.2.34 ADC_SOFTWARE_START

```
#define ADC_SOFTWARE_START ((uint32_t)ADC_CR2_EXTSEL + 1)
```

6.2.2.35 ADC_SQR1

```
#define ADC_SQR1(
    _NbrOfConversion_ ) (((_NbrOfConversion_) - (uint8_t)1) << 20)
```


6.2.2.36 ADC_SQR1_RK

```
#define ADC_SQR1_RK(  
    _CHANNELNB_,  
    _RANKNB_ ) (((uint32_t)((uint16_t)(_CHANNELNB_))) << (5 * ((_RANKNB_) - 13)))
```

6.2.2.37 ADC_SQR2_RK

```
#define ADC_SQR2_RK(  
    _CHANNELNB_,  
    _RANKNB_ ) (((uint32_t)((uint16_t)(_CHANNELNB_))) << (5 * ((_RANKNB_) - 7)))
```

6.2.2.38 ADC_SQR3_RK

```
#define ADC_SQR3_RK(  
    _CHANNELNB_,  
    _RANKNB_ ) (((uint32_t)((uint16_t)(_CHANNELNB_))) << (5 * ((_RANKNB_) - 1)))
```

6.2.2.39 ADC_STAB_DELAY_US

```
#define ADC_STAB_DELAY_US ((uint32_t) 3)
```

6.2.2.40 GPIO_MODE

```
#define GPIO_MODE ((uint32_t)0x00000003)
```

6.2.2.41 GPIO_NOPULL

```
#define GPIO_NOPULL ((uint32_t)0x00000000)  
No Pull-up or Pull-down activation
```

6.2.2.42 GPIO_PULLDOWN

```
#define GPIO_PULLDOWN ((uint32_t)0x00000002)  
Pull-down activation
```

6.2.2.43 GPIO_PULLUP

```
#define GPIO_PULLUP ((uint32_t)0x00000001)  
Pull-up activation
```

6.2.2.44 GPIO_SPEED_FAST

```
#define GPIO_SPEED_FAST ((uint32_t)0x00000002)  
Fast speed
```

6.2.2.45 GPIO_SPEED_HIGH

```
#define GPIO_SPEED_HIGH ((uint32_t)0x00000003)  
High speed
```

6.2.2.46 GPIO_SPEED_LOW

```
#define GPIO_SPEED_LOW ((uint32_t)0x00000000)  
Low speed
```

6.2.2.47 GPIO_SPEED_MEDIUM

```
#define GPIO_SPEED_MEDIUM ((uint32_t)0x00000001)
Medium speed
```

6.2.2.48 GPIOA_BASE

```
#define GPIOA_BASE (AHB1PERIPH_BASE + 0x0000)
```

6.2.2.49 RCC_ADC1_CLK_ENABLE

```
#define RCC_ADC1_CLK_ENABLE( )
Value:
```

```
do { \
    __IO uint32_t tmpreg; \
    SET_BIT(RCC->APB2ENR, RCC_APB2ENR_ADC1EN);\
    /* Delay after an RCC peripheral clock enabling */ \
    tmpreg = READ_BIT(RCC->APB2ENR, RCC_APB2ENR_ADC1EN);\
    UNUSED(tmpreg); \
} while(0)
```

6.2.2.50 RCC_GPIOA_CLK_ENABLE

```
#define RCC_GPIOA_CLK_ENABLE( )
Value:
```

```
do { \
    __IO uint32_t tmpreg; \
    SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOAEN);\
    /* Delay after an RCC peripheral clock enabling */ \
    tmpreg = READ_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOAEN);\
    UNUSED(tmpreg); \
} while(0)
```

6.2.2.51 RCC_GPIOB_CLK_ENABLE

```
#define RCC_GPIOB_CLK_ENABLE( )
Value:
```

```
do { \
    __IO uint32_t tmpreg; \
    SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOBEN);\
    /* Delay after an RCC peripheral clock enabling */ \
    tmpreg = READ_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOBEN);\
    UNUSED(tmpreg); \
} while(0)
```

6.2.2.52 RCC_GPIOC_CLK_ENABLE

```
#define RCC_GPIOC_CLK_ENABLE( )
Value:
```

```
do { \
    __IO uint32_t tmpreg; \
    SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOCEN);\
    /* Delay after an RCC peripheral clock enabling */ \
    tmpreg = READ_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOCEN);\
    UNUSED(tmpreg); \
} while(0)
```

6.2.2.53 STM_MODE_ANALOG

```
#define STM_MODE_ANALOG (5)
```

6.2.2.54 STM_PIN

```
#define STM_PIN(
    X ) ((uint32_t)(X) & 0xF)
```

6.2.2.55 STM_PIN_AFNUM

```
#define STM_PIN_AFNUM(  
    X ) (((X) >> 7) & 0x0F)
```

6.2.2.56 STM_PIN_CHANNEL

```
#define STM_PIN_CHANNEL(  
    X ) (((X) >> 11) & 0x0F)
```

6.2.2.57 STM_PIN_DATA_EXT

```
#define STM_PIN_DATA_EXT(  
    MODE,  
    PUPD,  
    AFNUM,  
    CHANNEL,  
    INVERTED ) ((int)(((INVERTED & 0x01) << 15) | ((CHANNEL & 0x0F) << 11) | ((AFNUM  
& 0x0F) << 7) | ((PUPD & 0x07) << 4) | ((MODE & 0x0F) << 0)))
```

6.2.2.58 STM_PIN_MODE

```
#define STM_PIN_MODE(  
    X ) (((X) >> 0) & 0x0F)
```

6.2.2.59 STM_PIN_PUPD

```
#define STM_PIN_PUPD(  
    X ) (((X) >> 4) & 0x07)
```

6.2.2.60 STM_PORT

```
#define STM_PORT(  
    X ) (((uint32_t)(X) >> 4) & 0xF)
```

6.2.2.61 UNUSED

```
#define UNUSED(  
    x ) ((void)(x))
```

6.2.3 Enumeration Type Documentation**6.2.3.1 ADCName**

enum [ADCName](#)

Enumerator

ADC↔ _1	
------------	--

6.2.3.2 HAL_ADC_StateTypeDef

enum [HAL_ADC_StateTypeDef](#)

Enumerator

HAL_ADC_STATE_RESET	ADC not yet initialized or disabled
HAL_ADC_STATE_READY	ADC peripheral ready for use

Enumerator

HAL_ADC_STATE_BUSY	An internal process is ongoing
HAL_ADC_STATE_BUSY_REG	Regular conversion is ongoing
HAL_ADC_STATE_BUSY_INJ	Injected conversion is ongoing
HAL_ADC_STATE_BUSY_INJ_REG	Injected and regular conversion are ongoing
HAL_ADC_STATE_TIMEOUT	Timeout state
HAL_ADC_STATE_ERROR	ADC state error
HAL_ADC_STATE_EOC	Conversion is completed
HAL_ADC_STATE_EOC_REG	Regular conversion is completed
HAL_ADC_STATE_EOC_INJ	Injected conversion is completed
HAL_ADC_STATE_EOC_INJ_REG	Injected and regular conversion are completed
HAL_ADC_STATE_AWD	ADC state analog watchdog

6.2.3.3 HAL_LockTypeDef

```
enum HAL_LockTypeDef
```

Enumerator

HAL_UNLOCKED	
HAL_LOCKED	

6.2.3.4 PortName

```
enum PortName
```

Enumerator

PortA	
PortB	
PortC	
PortD	
PortE	
PortH	

6.2.4 Function Documentation

6.2.4.1 _ADC_ConfigChannel()

```
void _ADC_ConfigChannel (
    ADC_HandleTypeDef * hadc,
    ADC_ChannelConfTypeDef * sConfig )
```

6.2.4.2 _ADC_GetValue()

```
uint32_t _ADC_GetValue (
    ADC_HandleTypeDef * hadc )
```

6.2.4.3 _ADC_Init()

```
void _ADC_Init (
    ADC_HandleTypeDef * hadc )
```

6.2.4.4 `_ADC_PollForConversion()`

```
int _ADC_PollForConversion (
    ADC_HandleTypeDef * hadc,
    uint32_t Timeout )
```

6.2.4.5 `_adc_read()`

```
uint16_t _adc_read (
    analogin_s * obj )
```

Reads the current value of the ADC.

Returns

Potential of the pin, relative to ground.

6.2.4.6 `_ADC_Start()`

```
void _ADC_Start (
    ADC_HandleTypeDef * hadc )
```

6.2.4.7 `_GPIO_Init()`

```
void _GPIO_Init (
    GPIO_TypeDef * GPIOx,
    GPIO_InitTypeDef * GPIO_Init )
```

6.2.4.8 `adc_init()`

```
void adc_init (
    Pin pin )
```

6.2.4.9 `adc_read()`

```
uint16_t adc_read (
    Pin pin )
```

6.2.4.10 `analogin_init()`

```
void analogin_init (
    analogin_s * obj,
    Pin pin )
```

6.2.4.11 `pin_function()`

```
void pin_function (
    Pin pin,
    int data )
```

6.2.4.12 `pinmap_find_peripheral()`

```
uint32_t pinmap_find_peripheral (
    Pin pin )
```

6.2.4.13 `pinmap_function()`

```
uint32_t pinmap_function (
    Pin pin )
```

Initializes the analogue to digital converter, and configures the appropriate GPIO pin.

6.2.4.14 pinmap_peripheral()

```
uint32_t pinmap_peripheral (
    Pin pin )
```

6.2.4.15 pinmap_pinout()

```
void pinmap_pinout (
    Pin pin )
```

6.3 adc.h

[Go to the documentation of this file.](#)

```
00001
00006 #ifndef ADC_H
00007 #define ADC_H
00008 #include "stm32f4xx_adc.h"
00009
00010 #define ADC1_BASE (APB2PERIPH_BASE + 0x2000)
00011 #define STM_PIN_DATA_EXT(MODE, PUPD, AFNUM, CHANNEL, INVERTED) ((int)(((INVERTED & 0x01) << 15) |
    ((CHANNEL & 0x0F) << 11) | ((AFNUM & 0x0F) << 7) | ((PUPD & 0x07) << 4) | ((MODE & 0x0F) << 0)))
00012 #define STM_MODE_ANALOG (5)
00013 #define STM_PIN_CHANNEL(X) (((X) >> 11) & 0x0F)
00014 #define STM_PIN_MODE(X) (((X) >> 0) & 0x0F)
00015 #define STM_PIN_PUPD(X) (((X) >> 4) & 0x07)
00016 #define STM_PIN_AFNUM(X) (((X) >> 7) & 0x0F)
00017 #define STM_PORT(X) (((uint32_t)(X) >> 4) & 0xF)
00018 #define STM_PIN(X) ((uint32_t)(X) & 0xF)
00019 #define GPIO_NOPULL ((uint32_t)0x00000000)
00020 #define GPIO_PULLUP ((uint32_t)0x00000001)
00021 #define GPIO_PULLDOWN ((uint32_t)0x00000002)
00022 #define UNUSED(x) ((void)(x))
00023 #define GPIOA_BASE (AHB1PERIPH_BASE + 0x0000)
00024 #define GPIO_SPEED_LOW ((uint32_t)0x00000000)
00025 #define GPIO_SPEED_MEDIUM ((uint32_t)0x00000001)
00026 #define GPIO_SPEED_FAST ((uint32_t)0x00000002)
00027 #define GPIO_SPEED_HIGH ((uint32_t)0x00000003)
00028 #define GPIO_MODE ((uint32_t)0x00000003)
00029 #define ADC_CR1_SCANCONV(_SCANCONV_MODE_) ((_SCANCONV_MODE_) << 8)
00030 #define ADC_SOFTWARE_START ((uint32_t)ADC_CR2_EXTSEL + 1)
00031 #define ADC_CR2_CONTINUOUS(_CONTINUOUS_MODE_) ((_CONTINUOUS_MODE_) << 1)
00032 #define ADC_CR1_DISCONTINUOUS(_NBR_DISCONTINUOUSCONV_) (((_NBR_DISCONTINUOUSCONV_) - 1) <<
    POSITION_VAL(ADC_CR1_DISCNUM))
00033 #define ADC_SQR1(_NbrOfConversion_) (((_NbrOfConversion_) - (uint8_t)1) << 20)
00034 #define ADC_CR2_DMAContReq(_DMAContReq_MODE_) ((_DMAContReq_MODE_) << 9)
00035 #define ADC_CR2_EOCSelection(_EOCSelection_MODE_) ((_EOCSelection_MODE_) << 10)
00036 #define ADC_SMPR1(_SAMPLETIME_, _CHANNELNB_) ((_SAMPLETIME_) << (3 *
    ((uint32_t)((uint16_t)(_CHANNELNB_)) - 10)))
00037 #define ADC_SMPR2(_SAMPLETIME_, _CHANNELNB_) ((_SAMPLETIME_) << (3 *
    ((uint32_t)((uint16_t)(_CHANNELNB_)))))
00038 #define ADC_SQR3_RK(_CHANNELNB_, _RANKNB_) (((uint32_t)((uint16_t)(_CHANNELNB_))) << (5 * ((_RANKNB_) -
    1)))
00039 #define ADC_SQR2_RK(_CHANNELNB_, _RANKNB_) (((uint32_t)((uint16_t)(_CHANNELNB_))) << (5 * ((_RANKNB_) -
    7)))
00040 #define ADC_SQR1_RK(_CHANNELNB_, _RANKNB_) (((uint32_t)((uint16_t)(_CHANNELNB_))) << (5 * ((_RANKNB_) -
    13)))
00041 #define ADC_CHANNEL_VBAT ((uint32_t)ADC_CHANNEL_18)
00042 #define ADC_CHANNEL_VREFINT ((uint32_t)ADC_CHANNEL_17)
00043 #define ADC_CHANNEL_TEMPSENSOR ((uint32_t)ADC_CHANNEL_16)
00044 #define ADC_STAB_DELAY_US ((uint32_t)3)
00045 #define _IS_BIT_CLR(REG, BIT) (((REG) & (BIT)) == RESET)
00046 #define _IS_BIT_SET(REG, BIT) (((REG) & (BIT)) != RESET)
00047 #define _ADC_GET_FLAG(__HANDLE__, __FLAG__) ((((__HANDLE__)->Instance->SR) & (__FLAG__)) ==
    (__FLAG__))
00048
00049 #define RCC_GPIOA_CLK_ENABLE() do { \
00050     __IO uint32_t tmpreg; \
00051     SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOAEN); \
00052     /* Delay after an RCC peripheral clock enabling */ \
00053     tmpreg = READ_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOAEN); \
00054     UNUSED(tmpreg); \
00055 } while(0)
00056 #define RCC_GPIOB_CLK_ENABLE() do { \
00057     __IO uint32_t tmpreg; \
00058     SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOBEN); \
00059     /* Delay after an RCC peripheral clock enabling */ \
00060     tmpreg = READ_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOBEN); \
00061     UNUSED(tmpreg); \
00062 } while(0)
00063 #define RCC_GPIOC_CLK_ENABLE() do { \
00064     __IO uint32_t tmpreg; \
```

```

00065 SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOCEN);\
00066 /* Delay after an RCC peripheral clock enabling */ \
00067 tmpreg = READ_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOCEN);\
00068 UNUSED(tmpreg); \
00069 } while(0)
00070
00071 #define RCC_ADC1_CLK_ENABLE() do { \
00072     __IO uint32_t tmpreg; \
00073     SET_BIT(RCC->APB2ENR, RCC_APB2ENR_ADC1EN);\
00074     /* Delay after an RCC peripheral clock enabling */ \
00075     tmpreg = READ_BIT(RCC->APB2ENR, RCC_APB2ENR_ADC1EN);\
00076     UNUSED(tmpreg); \
00077 } while(0)
00078
00079 #define ADC_CHANNEL_0 ((uint32_t)0x00000000)
00080 #define ADC_CHANNEL_1 ((uint32_t)ADC_CR1_AWDCH_0)
00081 #define ADC_CHANNEL_2 ((uint32_t)ADC_CR1_AWDCH_1)
00082 #define ADC_CHANNEL_3 ((uint32_t)(ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))
00083 #define ADC_CHANNEL_4 ((uint32_t)ADC_CR1_AWDCH_2)
00084 #define ADC_CHANNEL_5 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_0))
00085 #define ADC_CHANNEL_6 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1))
00086 #define ADC_CHANNEL_7 ((uint32_t)(ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))
00087 #define ADC_CHANNEL_8 ((uint32_t)ADC_CR1_AWDCH_3)
00088 #define ADC_CHANNEL_9 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_0))
00089 #define ADC_CHANNEL_10 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_1))
00090 #define ADC_CHANNEL_11 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_1 | ADC_CR1_AWDCH_0))
00091 #define ADC_CHANNEL_12 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2))
00092 #define ADC_CHANNEL_13 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_0))
00093 #define ADC_CHANNEL_14 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1))
00094 #define ADC_CHANNEL_15 ((uint32_t)(ADC_CR1_AWDCH_3 | ADC_CR1_AWDCH_2 | ADC_CR1_AWDCH_1 |
    ADC_CR1_AWDCH_0))
00095 #define ADC_CHANNEL_16 ((uint32_t)ADC_CR1_AWDCH_4)
00096 #define ADC_CHANNEL_17 ((uint32_t)(ADC_CR1_AWDCH_4 | ADC_CR1_AWDCH_0))
00097 #define ADC_CHANNEL_18 ((uint32_t)(ADC_CR1_AWDCH_4 | ADC_CR1_AWDCH_1))
00098
00099
00100 typedef enum
00101 {
00102     HAL_UNLOCKED = 0x00,
00103     HAL_LOCKED = 0x01
00104 } HAL_LockTypeDef;
00105
00106 typedef enum
00107 {
00108     HAL_ADC_STATE_RESET = 0x00,
00109     HAL_ADC_STATE_READY = 0x01,
00110     HAL_ADC_STATE_BUSY = 0x02,
00111     HAL_ADC_STATE_BUSY_REG = 0x12,
00112     HAL_ADC_STATE_BUSY_INJ = 0x22,
00113     HAL_ADC_STATE_BUSY_INJ_REG = 0x32,
00114     HAL_ADC_STATE_TIMEOUT = 0x03,
00115     HAL_ADC_STATE_ERROR = 0x04,
00116     HAL_ADC_STATE_EOC = 0x05,
00117     HAL_ADC_STATE_EOC_REG = 0x15,
00118     HAL_ADC_STATE_EOC_INJ = 0x25,
00119     HAL_ADC_STATE_EOC_INJ_REG = 0x35,
00120     HAL_ADC_STATE_AWD = 0x06
00121 } HAL_ADC_StateTypeDef;
00122
00123
00124 typedef struct
00125 {
00126     uint32_t ClockPrescaler;
00127     uint32_t Resolution;
00128     uint32_t DataAlign;
00129     uint32_t ScanConvMode;
00130     uint32_t EOCSelection;
00131     uint32_t ContinuousConvMode;
00132     uint32_t DMAContinuousRequests;
00133     uint32_t NbrOfConversion;
00134     uint32_t DiscontinuousConvMode;
00135     uint32_t NbrOfDiscConversion;
00136     uint32_t ExternalTrigConv;
00137     uint32_t ExternalTrigConvEdge;
00138 } _ADC_InitTypeDef;
00139
00140 typedef struct
00141 {
00142     ADC_TypeDef *Instance;
00143     _ADC_InitTypeDef Init;
00144     __IO uint32_t NbrOfCurrentConversionRank;
00145     //DMA_HandleTypeDef *DMA_Handle; /*!< Pointer DMA Handler */
00146     HAL_LockTypeDef Lock;
00147     __IO HAL_ADC_StateTypeDef State;
00148     __IO uint32_t ErrorCode;
00149 } ADC_HandleTypeDef;
00150
00151

```

```

00183 typedef struct
00184 {
00185     uint32_t Pin;
00187     uint32_t Mode;
00189     uint32_t Pull;
00191     uint32_t Speed;
00193     uint32_t Alternate;
00195 }GPIO_InitTypeDef;
00196
00197 typedef enum {
00198     PortA = 0,
00199     PortB = 1,
00200     PortC = 2,
00201     PortD = 3,
00202     PortE = 4,
00203     PortH = 7
00204 } PortName;
00205
00206 typedef struct {
00207     Pin pin;
00208     int peripheral;
00209     int function;
00210 } PinMap;
00211
00212
00213
00214
00215 typedef enum {
00216     ADC_1 = (int)ADC1_BASE
00217 } ADCName;
00218
00219
00220 typedef struct {
00221     ADCName adc;
00222     Pin pin;
00223     uint8_t channel;
00224 }analogin_s ;
00225
00226 typedef struct
00227 {
00228     uint32_t Channel;
00230     uint32_t Rank;
00232     uint32_t SamplingTime;
00234     uint32_t Offset;
00235 }ADC_ChannelConfTypeDef;
00236
00237
00241 uint32_t pinmap_function(Pin pin);
00242 void analogin_init(analogin_s *obj, Pin pin);
00243
00244 uint32_t pinmap_peripheral(Pin pin);
00245 uint32_t pinmap_find_peripheral(Pin pin);
00246 void adc_init(Pin pin);
00247 void pinmap_pinout(Pin pin);
00248 void pin_function(Pin pin, int data);
00249 void _GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init);
00250 void _ADC_Init(ADC_HandleTypeDef* hadc);
00251 static void local_ADC_Init(ADC_HandleTypeDef* hadc);
00252 void _ADC_ConfigChannel(ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef* sConfig);
00253 void _ADC_Start(ADC_HandleTypeDef* hadc);
00254 int _ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t Timeout);
00255 uint32_t _ADC_GetValue(ADC_HandleTypeDef* hadc);
00256
00257
00258
00262 uint16_t _adc_read(analogin_s *obj);
00263 uint16_t adc_read(Pin pin);
00264
00265 #endif // ADC_H

```

6.4 drivers/comparator.c File Reference

```

#include <platform.h>
#include <comparator.h>
#include <stdlib.h>

```

Functions

- void [comparator_init](#) (void)

Initializes the internal comparator.

- int [comparator_read](#) (void)

6.4.1 Function Documentation

6.4.1.1 [comparator_init\(\)](#)

```
void comparator_init (
    void )
```

Initializes the internal comparator.

6.4.1.2 [comparator_read\(\)](#)

```
int comparator_read (
    void )
```

6.5 drivers/comparator.h File Reference

Exposes functions of an internal comparator.

```
#include <adc.h>
```

Enumerations

- enum [ComparatorTriggerMode](#) { [CompNone](#) , [CompRising](#) , [CompFalling](#) , [CompBoth](#) }

Functions

- void [comparator_init](#) (void)
Initializes the internal comparator.
- void [comparator_set_trigger](#) ([ComparatorTriggerMode](#) trig)
Reads the current value of the comparator.
- void [comparator_set_callback](#) (void(*callback)(int state))
Pass a callback to the API, which is executed during the interrupt handler.
- int [comparator_read](#) (void)

6.5.1 Detailed Description

Exposes functions of an internal comparator.

Copyright

ARM University Program © ARM Ltd 2014.

6.5.2 Enumeration Type Documentation

6.5.2.1 [ComparatorTriggerMode](#)

```
enum ComparatorTriggerMode
```

Defines the triggering mode of the comparator's interrupt.

Enumerator

CompNone	Disables the interrupt.
CompRising	Enables an interrupt on the falling edge.
CompFalling	Enables an interrupt on the rising edge.
CompBoth	Enables an interrupt on both the rising and falling edges.

6.5.3 Function Documentation

6.5.3.1 comparator_init()

```
void comparator_init (
    void )
```

Initializes the internal comparator.

6.5.3.2 comparator_read()

```
int comparator_read (
    void )
```

6.5.3.3 comparator_set_callback()

```
void comparator_set_callback (
    void(*) (int state) callback )
```

Pass a callback to the API, which is executed during the interrupt handler.

See also

[comparator_set_trigger](#) to configure and enable the interrupt.

Parameters

<i>callback</i>	Callback function.
-----------------	--------------------

6.5.3.4 comparator_set_trigger()

```
void comparator_set_trigger (
    ComparatorTriggerMode trig )
```

Reads the current value of the comparator.

Returns

Output value of the comparator.

Configures the event which will cause an interrupt.

Parameters

<i>trig</i>	New triggering mode.
-------------	----------------------

6.6 comparator.h

[Go to the documentation of this file.](#)

```
00001
00006 #include <adc.h>
00007 #ifndef COMPARATOR_H
00008 #define COMPARATOR_H
00009
00010
00012 typedef enum {
00013     CompNone,
00014     CompRising,
00015     CompFalling,
00016     CompBoth
00017 } ComparatorTriggerMode;
00018
00019
00021 void comparator_init(void);
00022
00026 //int comparator_read(void);
```

```

00027
00031 void comparator_set_trigger(ComparatorTriggerMode trig);
00032
00040 void comparator_set_callback(void (*callback)(int state));
00041
00042 int comparator_read(void);
00043
00044 #endif // COMPARATOR_H

```

6.7 drivers/gpio.c File Reference

```

#include <platform.h>
#include <gpio.h>

```

Functions

- void [gpio_toggle](#) ([Pin](#) pin)
Toggles a GPIO pin's output. A pin which is currently high is set low and a pin which is currently low is set high.
- void [gpio_set](#) ([Pin](#) pin, int value)
Sets a pin to the specified logic level.
- int [gpio_get](#) ([Pin](#) pin)
Get the current logic level of a GPIO pin. If the pin is high, this function will return a 1, else it will return 0.
- void [gpio_set_range](#) ([Pin](#) pin_base, int count, int value)
Sets a range of sequential pins to the specified value.
- unsigned int [gpio_get_range](#) ([Pin](#) pin_base, int count)
Returns the value of a range of sequential pins.
- void [gpio_set_mode](#) ([Pin](#) pin, [PinMode](#) mode)
Configures the output mode of a GPIO pin.
- void [gpio_set_trigger](#) ([Pin](#) pin, [TriggerMode](#) trig)
Configures the event which will cause an interrupt on a specified pin.
- void [gpio_set_callback](#) ([Pin](#) pin, void(*callback)(int status))
Passes a callback function to the api which is called during the port's relevant interrupt.
- void [EXTI0_IRQHandler](#) (void)
- void [EXTI1_IRQHandler](#) (void)
- void [EXTI2_IRQHandler](#) (void)
- void [EXTI3_IRQHandler](#) (void)
- void [EXTI4_IRQHandler](#) (void)
- void [EXTI9_5_IRQHandler](#) (void)
- void [EXTI15_10_IRQHandler](#) (void)

Variables

- uint32_t [IRQ_status](#)
- uint32_t [IRQ_port_num](#)
- uint32_t [IRQ_pin_index](#)
- uint32_t [EXTI_port_set](#)
- uint32_t [prioritygroup](#)
- uint32_t [priority](#)

6.7.1 Function Documentation

6.7.1.1 EXTI0_IRQHandler()

```

void EXTI0_IRQHandler (
    void )

```

6.7.1.2 EXTI15_10_IRQHandler()

```
void EXTI15_10_IRQHandler (
    void )
```

6.7.1.3 EXTI1_IRQHandler()

```
void EXTI1_IRQHandler (
    void )
```

6.7.1.4 EXTI2_IRQHandler()

```
void EXTI2_IRQHandler (
    void )
```

6.7.1.5 EXTI3_IRQHandler()

```
void EXTI3_IRQHandler (
    void )
```

6.7.1.6 EXTI4_IRQHandler()

```
void EXTI4_IRQHandler (
    void )
```

6.7.1.7 EXTI9_5_IRQHandler()

```
void EXTI9_5_IRQHandler (
    void )
```

6.7.1.8 gpio_get()

```
int gpio_get (
    Pin pin )
```

Get the current logic level of a GPIO pin. If the pin is high, this function will return a 1, else it will return 0.

Parameters

<i>pin</i>	Pin to read.
------------	--------------

Returns

The logic level of the GPIO pin (0 if low, 1 if high).

6.7.1.9 gpio_get_range()

```
unsigned int gpio_get_range (
    Pin pin_base,
    int count )
```

Returns the value of a range of sequential pins.

Parameters

<i>pin_base</i>	Starting pin.
<i>count</i>	Number of pins to set.

Returns

Value of the pins.

6.7.1.10 gpio_set()

```
void gpio_set (
    Pin pin,
    int value )
```

Sets a pin to the specified logic level.

Parameters

<i>pin</i>	Pin to set.
<i>value</i>	New logic level of the pin (0 is low, otherwise high).

6.7.1.11 gpio_set_callback()

```
void gpio_set_callback (
    Pin pin,
    void(*) (int status) callback )
```

Passes a callback function to the api which is called during the port's relevant interrupt.

Warning

The pin argument specifies the port which will be interrupted on, not an individual pin. It is advised check the *status* variable to determine which pin caused the interrupt.

See also

[gpio_set_trigger](#) to configure and enable the interrupt.

Parameters

<i>pin</i>	Pin which specifies the port to use.
<i>callback</i>	Callback function.

6.7.1.12 gpio_set_mode()

```
void gpio_set_mode (
    Pin pin,
    PinMode mode )
```

Configures the output mode of a GPIO pin.

Used to set the GPIO as an input, output, and configure the possible pull-up or pull-down resistors.

Parameters

<i>pin</i>	Pin to set.
<i>mode</i>	New output mode of the pin.

6.7.1.13 gpio_set_range()

```
void gpio_set_range (
    Pin pin_base,
```

```
int count,
int value )
```

Sets a range of sequential pins to the specified value.

Parameters

<i>pin_base</i>	Starting pin.
<i>count</i>	Number of pins to set.
<i>value</i>	New value of the pins.

6.7.1.14 gpio_set_trigger()

```
void gpio_set_trigger (
    Pin pin,
    TriggerMode trig )
```

Configures the event which will cause an interrupt on a specified pin.

Parameters

<i>pin</i>	Pin to trigger off.
<i>trig</i>	New triggering mode for the pin.

6.7.1.15 gpio_toggle()

```
void gpio_toggle (
    Pin pin )
```

Toggles a GPIO pin's output. A pin which is currently high is set low and a pin which is currently low is set high.

Parameters

<i>pin</i>	Pin to toggle.
------------	----------------

6.7.2 Variable Documentation

6.7.2.1 EXTI_port_set

```
uint32_t EXTI_port_set
```

6.7.2.2 IRQ_pin_index

```
uint32_t IRQ_pin_index
```

6.7.2.3 IRQ_port_num

```
uint32_t IRQ_port_num
```

6.7.2.4 IRQ_status

```
uint32_t IRQ_status
```

6.7.2.5 priority

```
uint32_t priority
```

6.7.2.6 prioritygroup

uint32_t prioritygroup

6.8 drivers/gpio.h File Reference

Implements general purpose I/O.

```
#include <platform.h>
```

Enumerations

- enum [PinMode](#) {
 [Reset](#) , [Input](#) , [Output](#) , [PullUp](#) ,
 [PullDown](#) }
- enum [TriggerMode](#) { [None](#) , [Rising](#) , [Falling](#) }

Functions

- void [gpio_toggle](#) ([Pin](#) pin)
Toggles a GPIO pin's output. A pin which is currently high is set low and a pin which is currently low is set high.
- void [gpio_set](#) ([Pin](#) pin, int value)
Sets a pin to the specified logic level.
- int [gpio_get](#) ([Pin](#) pin)
Get the current logic level of a GPIO pin. If the pin is high, this function will return a 1, else it will return 0.
- void [gpio_set_range](#) ([Pin](#) pin_base, int count, int value)
Sets a range of sequential pins to the specified value.
- unsigned int [gpio_get_range](#) ([Pin](#) pin_base, int count)
Returns the value of a range of sequential pins.
- void [gpio_set_mode](#) ([Pin](#) pin, [PinMode](#) mode)
Configures the output mode of a GPIO pin.
- void [gpio_set_trigger](#) ([Pin](#) pin, [TriggerMode](#) trig)
Configures the event which will cause an interrupt on a specified pin.
- void [gpio_set_callback](#) ([Pin](#) pin, void(*callback)(int status))
Passes a callback function to the api which is called during the port's relevant interrupt.

6.8.1 Detailed Description

Implements general purpose I/O.

Copyright

ARM University Program © ARM Ltd 2014.

Exposes generic pin input / output controls. Use for any direct pin manipulation.

6.8.2 Enumeration Type Documentation

6.8.2.1 PinMode

enum [PinMode](#)

This enum describes the directional setup of a GPIO pin.

Enumerator

Reset	Resets the pin-mode to the default value.
Input	Sets the pin as an input with no pull-up or pull-down.

Enumerator

Output	Sets the pin as a low impedance output.
PullUp	Enables the internal pull-up resistor.
PullDown	Enables the internal pull-down resistor.

6.8.2.2 TriggerMode

enum `TriggerMode`

Defines the triggering mode of an interrupt.

Enumerator

None	Disables the interrupt.
Rising	Enables an interrupt on the falling edge.
Falling	Enables an interrupt on the rising edge.

6.8.3 Function Documentation**6.8.3.1 gpio_get()**

```
int gpio_get (
    Pin pin )
```

Get the current logic level of a GPIO pin. If the pin is high, this function will return a 1, else it will return 0.

Parameters

<i>pin</i>	Pin to read.
------------	--------------

Returns

The logic level of the GPIO pin (0 if low, 1 if high).

6.8.3.2 gpio_get_range()

```
unsigned int gpio_get_range (
    Pin pin_base,
    int count )
```

Returns the value of a range of sequential pins.

Parameters

<i>pin_base</i>	Starting pin.
<i>count</i>	Number of pins to set.

Returns

Value of the pins.

6.8.3.3 gpio_set()

```
void gpio_set (
    Pin pin,
    int value )
```


Sets a pin to the specified logic level.

Parameters

<i>pin</i>	Pin to set.
<i>value</i>	New logic level of the pin (0 is low, otherwise high).

6.8.3.4 gpio_set_callback()

```
void gpio_set_callback (
    Pin pin,
    void(*) (int status) callback )
```

Passes a callback function to the api which is called during the port's relevant interrupt.

Warning

The pin argument specifies the port which will be interrupted on, not an individual pin. It is advised check the *status* variable to determine which pin caused the interrupt.

See also

[gpio_set_trigger](#) to configure and enable the interrupt.

Parameters

<i>pin</i>	Pin which specifies the port to use.
<i>callback</i>	Callback function.

6.8.3.5 gpio_set_mode()

```
void gpio_set_mode (
    Pin pin,
    PinMode mode )
```

Configures the output mode of a GPIO pin.

Used to set the GPIO as an input, output, and configure the possible pull-up or pull-down resistors.

Parameters

<i>pin</i>	Pin to set.
<i>mode</i>	New output mode of the pin.

6.8.3.6 gpio_set_range()

```
void gpio_set_range (
    Pin pin_base,
    int count,
    int value )
```

Sets a range of sequential pins to the specified value.

Parameters

<i>pin_base</i>	Starting pin.
<i>count</i>	Number of pins to set.
<i>value</i>	New value of the pins.

6.8.3.7 gpio_set_trigger()

```
void gpio_set_trigger (
    Pin pin,
    TriggerMode trig )
```

Configures the event which will cause an interrupt on a specified pin.

Parameters

<i>pin</i>	Pin to trigger off.
<i>trig</i>	New triggering mode for the pin.

6.8.3.8 gpio_toggle()

```
void gpio_toggle (
    Pin pin )
```

Toggles a GPIO pin's output. A pin which is currently high is set low and a pin which is currently low is set high.

Parameters

<i>pin</i>	Pin to toggle.
------------	----------------

6.9 gpio.h

[Go to the documentation of this file.](#)

```
00001
00009 #ifndef PINS_H
00010 #define PINS_H
00011
00012 #include <platform.h>
00013
00015 typedef enum {
00016     Reset,
00017     Input,
00018     Output,
00019     PullUp,
00020     PullDown
00021 } PinMode;
00022
00024 typedef enum {
00025     None,
00026     Rising,
00027     Falling
00028 } TriggerMode;
00029
00035 void gpio_toggle(Pin pin);
00036
00041 void gpio_set(Pin pin, int value);
00042
00049 int gpio_get(Pin pin);
00050
00056 void gpio_set_range(Pin pin_base, int count, int value);
00057
00063 unsigned int gpio_get_range(Pin pin_base, int count);
00064
00073 void gpio_set_mode(Pin pin, PinMode mode);
00074
00081 void gpio_set_trigger(Pin pin, TriggerMode trig);
00082
00096 void gpio_set_callback(Pin pin, void (*callback)(int status));
00097
00098 #endif // PINS_H
00099
00100 // *****ARM University Program Copyright © ARM Ltd
    2016*****
```

6.10 drivers/i2c.c File Reference

```
#include <platform.h>
#include <stm32f4xx_i2c.h>
#include <STM32F4xx_RCC.h>
#include <STM32F4xx_GPIO.h>
```

Functions

- void [i2c_init](#) ()
Initialises the hardware I2C module, any relevant pins and enables the module.
- void [i2c_write](#) (uint8_t address, uint8_t *buffer, int buff_len)
Writes data to an I2C module.
- void [i2c_read](#) (uint8_t address, uint8_t *buffer, int buff_len)
Reads data from an I2C module.

6.10.1 Function Documentation

6.10.1.1 i2c_init()

```
void i2c_init (
    void )
```

Initialises the hardware I2C module, any relevant pins and enables the module.

6.10.1.2 i2c_read()

```
void i2c_read (
    uint8_t address,
    uint8_t * buffer,
    int buff_len )
```

Reads data from an I2C module.

Parameters

<i>address</i>	I2C address of the slave.
<i>buffer</i>	Data to be read.
<i>buff_len</i>	Number of bytes to read.

6.10.1.3 i2c_write()

```
void i2c_write (
    uint8_t address,
    uint8_t * buffer,
    int buff_len )
```

Writes data to an I2C module.

Parameters

<i>address</i>	I2C address of the slave.
<i>buffer</i>	Data to be sent.
<i>buff_len</i>	Number of bytes to send.

6.11 drivers/i2c.h File Reference

Controller for hardware I2C module, configured as a master.

```
#include <stdint.h>
```

Functions

- void [i2c_init](#) (void)
Initialises the hardware I2C module, any relevant pins and enables the module.
- void [i2c_write](#) (uint8_t address, uint8_t *buffer, int buff_len)
Writes data to an I2C module.
- void [i2c_read](#) (uint8_t address, uint8_t *buffer, int buff_len)
Reads data from an I2C module.

6.11.1 Detailed Description

Controller for hardware I2C module, configured as a master.

Copyright

ARM University Program © ARM Ltd 2014.

6.11.2 Function Documentation

6.11.2.1 i2c_init()

```
void i2c_init (
    void )
```

Initialises the hardware I2C module, any relevant pins and enables the module.

6.11.2.2 i2c_read()

```
void i2c_read (
    uint8_t address,
    uint8_t * buffer,
    int buff_len )
```

Reads data from an I2C module.

Parameters

<i>address</i>	I2C address of the slave.
<i>buffer</i>	Data to be read.
<i>buff_len</i>	Number of bytes to read.

6.11.2.3 i2c_write()

```
void i2c_write (
    uint8_t address,
    uint8_t * buffer,
    int buff_len )
```

Writes data to an I2C module.

Parameters

<i>address</i>	I2C address of the slave.
<i>buffer</i>	Data to be sent.
<i>buff_len</i>	Number of bytes to send.

6.12 i2c.h

[Go to the documentation of this file.](#)

```
00001
00007 #ifndef I2C_H
00008 #define I2C_H
00009 #include <stdint.h>
00010
00014 void i2c_init(void);
00015
00021 void i2c_write(uint8_t address, uint8_t *buffer, int buff_len);
00022
00028 void i2c_read(uint8_t address, uint8_t *buffer, int buff_len);
00029
00030 #endif //I2C_H
00031
00032 // *****ARM University Program Copyright © ARM Ltd
    2016*****
```

6.13 drivers/platform.h File Reference

```
#include <STM32F4xx.h>
```

Macros

- #define CLK_FREQ 8400000UL
- #define P_SW PC_13
- #define P_LED_R PA_5
- #define P_LED_G PA_6
- #define P_LED_B PA_7
- #define P_SW_UP PA_1
- #define P_SW_CR PA_4
- #define P_SW_DN PB_0
- #define P_SW_LT PC_1
- #define P_SW_RT PC_0
- #define P_DBG_ISR PC_8
- #define P_DBG_MAIN PC_6
- #define P_SPEAKER PB_6
- #define P_LCD_RS PA_9
- #define P_LCD_RW PA_8
- #define P_LCD_E PB_10
- #define P_LCD_DATA4 PB_4
- #define P_LCD_DATA5 PB_5
- #define P_LCD_DATA6 PB_3
- #define P_LCD_DATA7 PA_10
- #define P_IR PA_8
- #define P_SAMPLE PC_10
- #define P_PERIOD PC_12
- #define P_ADC PA_0
- #define P_CMP_PLUS PA_1
- #define P_CMP_NEG PA_0
- #define P_RX PA_3
- #define P_TX PA_2
- #define P_SCL PB_8
- #define P_SDA PB_9
- #define GET_PORT_INDEX(pin) ((pin) >> 16)
- #define GET_PIN_INDEX(pin) ((pin) & 0xFF)
- #define ADC_BITS 12
- #define ADC_MASK ((1u << ADC_BITS) - 1)

- `#define DAC_BITS 8`
- `#define DAC_MASK ((1u << DAC_BITS) - 1)`
- `#define LED_ON 1`
- `#define LED_OFF 0`
- `#define GET_PORT(pin) ((GPIO_TypeDef*)(AHB1PERIPH_BASE + 0x0400 * GET_PORT_INDEX(pin)))`

Enumerations

- `enum Pin {`
`PA_0 = (0 << 16) | 0, PA_1 = (0 << 16) | 1, PA_2 = (0 << 16) | 2, PA_3 = (0 << 16) | 3,`
`PA_4 = (0 << 16) | 4, PA_5 = (0 << 16) | 5, PA_6 = (0 << 16) | 6, PA_7 = (0 << 16) | 7,`
`PA_8 = (0 << 16) | 8, PA_9 = (0 << 16) | 9, PA_10 = (0 << 16) | 10, PA_11 = (0 << 16) | 11,`
`PA_12 = (0 << 16) | 12, PA_13 = (0 << 16) | 13, PA_14 = (0 << 16) | 14, PA_15 = (0 << 16) | 15,`
`PB_0 = (1 << 16) | 0, PB_1 = (1 << 16) | 1, PB_2 = (1 << 16) | 2, PB_3 = (1 << 16) | 3,`
`PB_4 = (1 << 16) | 4, PB_5 = (1 << 16) | 5, PB_6 = (1 << 16) | 6, PB_7 = (1 << 16) | 7,`
`PB_8 = (1 << 16) | 8, PB_9 = (1 << 16) | 9, PB_10 = (1 << 16) | 10, PB_12 = (1 << 16) | 12,`
`PB_13 = (1 << 16) | 13, PB_14 = (1 << 16) | 14, PB_15 = (1 << 16) | 15, PC_0 = (2 << 16) | 0,`
`PC_1 = (2 << 16) | 1, PC_2 = (2 << 16) | 2, PC_3 = (2 << 16) | 3, PC_4 = (2 << 16) | 4,`
`PC_5 = (2 << 16) | 5, PC_6 = (2 << 16) | 6, PC_7 = (2 << 16) | 7, PC_8 = (2 << 16) | 8,`
`PC_9 = (2 << 16) | 9, PC_10 = (2 << 16) | 10, PC_11 = (2 << 16) | 11, PC_12 = (2 << 16) | 12,`
`PC_13 = (2 << 16) | 13, PC_14 = (2 << 16) | 14, PC_15 = (2 << 16) | 15, PD_2 = (3 << 16) | 2,`
`PH_0 = (7 << 16) | 0, PH_1 = (7 << 16) | 1, NC = (int)0xFFFFFFFF }`

6.13.1 Macro Definition Documentation

6.13.1.1 ADC_BITS

```
#define ADC_BITS 12
```

6.13.1.2 ADC_MASK

```
#define ADC_MASK ((1u << ADC_BITS) - 1)
```

6.13.1.3 CLK_FREQ

```
#define CLK_FREQ 8400000UL
```

6.13.1.4 DAC_BITS

```
#define DAC_BITS 8
```

6.13.1.5 DAC_MASK

```
#define DAC_MASK ((1u << DAC_BITS) - 1)
```

6.13.1.6 GET_PIN_INDEX

```
#define GET_PIN_INDEX(  
    pin ) ((pin) & 0xFF)
```

6.13.1.7 GET_PORT

```
#define GET_PORT(  
    pin ) ((GPIO_TypeDef*)(AHB1PERIPH_BASE + 0x0400 * GET_PORT_INDEX(pin)))
```

6.13.1.8 GET_PORT_INDEX

```
#define GET_PORT_INDEX(  
    pin ) ((pin) >> 16)
```

6.13.1.9 LED_OFF

```
#define LED_OFF 0
```

6.13.1.10 LED_ON

```
#define LED_ON 1
```

6.13.1.11 P_ADC

```
#define P_ADC PA_0
```

6.13.1.12 P_CMP_NEG

```
#define P_CMP_NEG PA_0
```

6.13.1.13 P_CMP_PLUS

```
#define P_CMP_PLUS PA_1
```

6.13.1.14 P_DBG_ISR

```
#define P_DBG_ISR PC_8
```

6.13.1.15 P_DBG_MAIN

```
#define P_DBG_MAIN PC_6
```

6.13.1.16 P_IR

```
#define P_IR PA_8
```

6.13.1.17 P_LCD_DATA4

```
#define P_LCD_DATA4 PB_4
```

6.13.1.18 P_LCD_DATA5

```
#define P_LCD_DATA5 PB_5
```

6.13.1.19 P_LCD_DATA6

```
#define P_LCD_DATA6 PB_3
```

6.13.1.20 P_LCD_DATA7

```
#define P_LCD_DATA7 PA_10
```

6.13.1.21 P_LCD_E

```
#define P_LCD_E PB_10
```

6.13.1.22 P_LCD_RS

```
#define P_LCD_RS PA_9
```

6.13.1.23 P_LCD_RW

```
#define P_LCD_RW PA_8
```

6.13.1.24 P_LED_B

```
#define P_LED_B PA_7
```

6.13.1.25 P_LED_G

```
#define P_LED_G PA_6
```

6.13.1.26 P_LED_R

```
#define P_LED_R PA_5
```

6.13.1.27 P_PERIOD

```
#define P_PERIOD PC_12
```

6.13.1.28 P_RX

```
#define P_RX PA_3
```

6.13.1.29 P_SAMPLE

```
#define P_SAMPLE PC_10
```

6.13.1.30 P_SCL

```
#define P_SCL PB_8
```

6.13.1.31 P_SDA

```
#define P_SDA PB_9
```

6.13.1.32 P_SPEAKER

```
#define P_SPEAKER PB_6
```

6.13.1.33 P_SW

```
#define P_SW PC_13
```

6.13.1.34 P_SW_CR

```
#define P_SW_CR PA_4
```

6.13.1.35 P_SW_DN

```
#define P_SW_DN PB_0
```

6.13.1.36 P_SW_LT

```
#define P_SW_LT PC_1
```

6.13.1.37 P_SW_RT

```
#define P_SW_RT PC_0
```

6.13.1.38 P_SW_UP

```
#define P_SW_UP PA_1
```


6.13.1.39 P_TX

```
#define P_TX PA_2
```

6.13.2 Enumeration Type Documentation

6.13.2.1 Pin

```
enum Pin
```

Enumerator

PA_0	
PA_1	
PA_2	
PA_3	
PA_4	
PA_5	
PA_6	
PA_7	
PA_8	
PA_9	
PA_10	
PA_11	
PA_12	
PA_13	
PA_14	
PA_15	
PB_0	
PB_1	
PB_2	
PB_3	
PB_4	
PB_5	
PB_6	
PB_7	
PB_8	
PB_9	
PB_10	
PB_12	
PB_13	
PB_14	
PB_15	
PC_0	
PC_1	
PC_2	
PC_3	
PC_4	
PC_5	
PC_6	
PC_7	
PC_8	
PC_9	
PC_10	

Enumerator

PC_11	
PC_12	
PC_13	
PC_14	
PC_15	
PD_2	
PH_0	
PH_1	
NC	

6.14 platform.h

[Go to the documentation of this file.](#)

```

00001 #ifndef PLATFORM_H
00002 #define PLATFORM_H
00003
00004 /* TODO:
00005  * + Add an include with the CMSIS Peripheral Access Layer.
00006  *   The test project should now build and run.
00007  * + Set CLK_FREQ to the core clock frequency.
00008  * + Define all the platforms pins in the Pin enumeration.
00009  *   These do not need to be named Px_y, any form is allowed,
00010  *   for example PTXy.
00011  * + Ensure the GET_PORT_INDEX and GET_PIN_INDEX macros are
00012  *   consistent with the Pin enumeration.
00013  * + Set all the pin #defines to appropriate pins.
00014  * + Define DAC_BITS and ADC_BITS to the number of bits of the
00015  *   on-board ADC / DAC that is being used by the drivers.
00016  */
00017
00018 #include <STM32F4xx.h>
00019
00020 // Core CPU frequency.
00021 #define CLK_FREQ 8400000UL
00022
00023 typedef enum {
00024     PA_0 = (0 << 16) | 0,
00025     PA_1 = (0 << 16) | 1,
00026     PA_2 = (0 << 16) | 2,
00027     PA_3 = (0 << 16) | 3,
00028     PA_4 = (0 << 16) | 4,
00029     PA_5 = (0 << 16) | 5,
00030     PA_6 = (0 << 16) | 6,
00031     PA_7 = (0 << 16) | 7,
00032     PA_8 = (0 << 16) | 8,
00033     PA_9 = (0 << 16) | 9,
00034     PA_10 = (0 << 16) | 10,
00035     PA_11 = (0 << 16) | 11,
00036     PA_12 = (0 << 16) | 12,
00037     PA_13 = (0 << 16) | 13,
00038     PA_14 = (0 << 16) | 14,
00039     PA_15 = (0 << 16) | 15,
00040
00041     PB_0 = (1 << 16) | 0,
00042     PB_1 = (1 << 16) | 1,
00043     PB_2 = (1 << 16) | 2,
00044     PB_3 = (1 << 16) | 3,
00045     PB_4 = (1 << 16) | 4,
00046     PB_5 = (1 << 16) | 5,
00047     PB_6 = (1 << 16) | 6,
00048     PB_7 = (1 << 16) | 7,
00049     PB_8 = (1 << 16) | 8,
00050     PB_9 = (1 << 16) | 9,
00051     PB_10 = (1 << 16) | 10,
00052     PB_12 = (1 << 16) | 12,
00053     PB_13 = (1 << 16) | 13,
00054     PB_14 = (1 << 16) | 14,
00055     PB_15 = (1 << 16) | 15,
00056
00057     PC_0 = (2 << 16) | 0,
00058     PC_1 = (2 << 16) | 1,
00059     PC_2 = (2 << 16) | 2,
00060     PC_3 = (2 << 16) | 3,
00061     PC_4 = (2 << 16) | 4,

```

```

00062 PC_5 = (2 << 16) | 5,
00063 PC_6 = (2 << 16) | 6,
00064 PC_7 = (2 << 16) | 7,
00065 PC_8 = (2 << 16) | 8,
00066 PC_9 = (2 << 16) | 9,
00067 PC_10 = (2 << 16) | 10,
00068 PC_11 = (2 << 16) | 11,
00069 PC_12 = (2 << 16) | 12,
00070 PC_13 = (2 << 16) | 13,
00071 PC_14 = (2 << 16) | 14,
00072 PC_15 = (2 << 16) | 15,
00073
00074 PD_2 = (3 << 16) | 2,
00075
00076 PH_0 = (7 << 16) | 0,
00077 PH_1 = (7 << 16) | 1,
00078 // Not connected
00079 NC = (int)0xFFFFFFFF
00080 } Pin;
00081
00082 /* Pin definitions */
00083
00084 // Module 5: IntDemo, IntProjectReactionTime
00085 // Module 7: AnalogLabSignalGenerator
00086 // Module 8: TimerLabSignalGenerator
00087 // Push-button.
00088 #define P_SW PC_13
00089
00090 // Module 5, 6, 7, 8, 9
00091 // RGB LEDs.
00092 #define P_LED_R PA_5
00093 #define P_LED_G PA_6
00094 #define P_LED_B PA_7
00095
00096 // Module 6: GPIOProjectSlideWhistle, GPIOLabBasicUI
00097 // Joystick control.
00098 #define P_SW_UP PA_1
00099 #define P_SW_CR PA_4
00100 #define P_SW_DN PB_0
00101 #define P_SW_LT PC_1
00102 #define P_SW_RT PC_0
00103
00104 // Module 5: IntDemo
00105 // Debug signals.
00106 #define P_DBG_ISR PC_8
00107 #define P_DBG_MAIN PC_6
00108
00109 // Module 6: GPIOProjectSlideWhistle
00110 // Speaker driven with GPIO.
00111 #define P_SPEAKER PB_6
00112
00113 // Module 6: GPIOLabBasicUI
00114 // Module 8: TimerProjectClock
00115 // Module 9: SerialDemoUART, SerialProjectGPSSpeedometer
00116 // LCD control.
00117 #define P_LCD_RS PA_9
00118 #define P_LCD_RW PA_8
00119 #define P_LCD_E PB_10
00120 #define P_LCD_DATA4 PB_4
00121 #define P_LCD_DATA5 PB_5
00122 #define P_LCD_DATA6 PB_3
00123 #define P_LCD_DATA7 PA_10
00124
00125 // Module 7, AnalogProject
00126 // IR LED.
00127 #define P_IR PA_8
00128
00129 // Module 8 Timers
00130 #define P_SAMPLE PC_10
00131 #define P_PERIOD PC_12
00132
00133 // Other pins (for documentation).
00134 #define P_ADC PA_0
00135 // #define P_DAC PA
00136 #define P_CMP_PLUS PA_1
00137 #define P_CMP_NEG PA_0
00138 #define P_RX PA_3
00139 #define P_TX PA_2
00140 #define P_SCL PB_8
00141 #define P_SDA PB_9
00142
00143 /* Other useful macros */
00144
00145 #define GET_PORT_INDEX(pin) ((pin) >> 16)
00146 #define GET_PIN_INDEX(pin) ((pin) & 0xFF)
00147
00148 #define ADC_BITS 12

```

```

00149 #define ADC_MASK ((1u << ADC_BITS) - 1)
00150 #define DAC_BITS 8
00151 #define DAC_MASK ((1u << DAC_BITS) - 1)
00152
00153 // LEDs are active high or low?
00154 #define LED_ON 1
00155 #define LED_OFF 0
00156
00157 #define GET_PORT(pin) ((GPIO_TypeDef*)(AHB1PERIPH_BASE + 0x0400 * GET_PORT_INDEX(pin)))
00158
00159 #endif
00160
00161 // *****ARM University Program Copyright ARM Ltd
    2016*****

```

6.15 drivers/stm32f4xx_adc.c File Reference

This file provides firmware functions to manage the following functionalities of the Analog to Digital Convertor (ADC) peripheral:

```

#include "stm32f4xx_adc.h"
#include "stm32f4xx_rcc.h"

```

Macros

- #define [CR1_DISCNUM_RESET](#) ((uint32_t)0xFFFF1FFF)
- #define [CR1_AWDCH_RESET](#) ((uint32_t)0xFFFFFFE0)
- #define [CR1_AWDMode_RESET](#) ((uint32_t)0xFF3FFDFF)
- #define [CR1_CLEAR_MASK](#) ((uint32_t)0xFCFFFEFF)
- #define [CR2_EXTEN_RESET](#) ((uint32_t)0xCFFFFFFF)
- #define [CR2_JEXTEN_RESET](#) ((uint32_t)0xFFCFFFFFFF)
- #define [CR2_JEXTSEL_RESET](#) ((uint32_t)0xFFF0FFFF)
- #define [CR2_CLEAR_MASK](#) ((uint32_t)0xC0FFF7FD)
- #define [SQR3_SQ_SET](#) ((uint32_t)0x0000001F)
- #define [SQR2_SQ_SET](#) ((uint32_t)0x0000001F)
- #define [SQR1_SQ_SET](#) ((uint32_t)0x0000001F)
- #define [SQR1_L_RESET](#) ((uint32_t)0xFF0FFFFF)
- #define [JSQR_JSQ_SET](#) ((uint32_t)0x0000001F)
- #define [JSQR_JL_SET](#) ((uint32_t)0x00300000)
- #define [JSQR_JL_RESET](#) ((uint32_t)0xFFCFFFFFFF)
- #define [SMPR1_SMP_SET](#) ((uint32_t)0x00000007)
- #define [SMPR2_SMP_SET](#) ((uint32_t)0x00000007)
- #define [JDR_OFFSET](#) ((uint8_t)0x28)
- #define [CDR_ADDRESS](#) ((uint32_t)0x40012308)
- #define [CR_CLEAR_MASK](#) ((uint32_t)0xFFFC30E0)

Functions

- void [ADC_DeInit](#) (void)
Deinitializes all ADCs peripherals registers to their default reset values.
- void [ADC_Init](#) (ADC_TypeDef *ADCx, [ADC_InitTypeDef](#) *ADC_InitStruct)
Initializes the ADCx peripheral according to the specified parameters in the ADC_InitStruct.
- void [ADC_StructInit](#) ([ADC_InitTypeDef](#) *ADC_InitStruct)
Fills each ADC_InitStruct member with its default value.
- void [ADC_CommonInit](#) ([ADC_CommonInitTypeDef](#) *ADC_CommonInitStruct)
Initializes the ADCs peripherals according to the specified parameters in the ADC_CommonInitStruct.
- void [ADC_CommonStructInit](#) ([ADC_CommonInitTypeDef](#) *ADC_CommonInitStruct)
Fills each ADC_CommonInitStruct member with its default value.
- void [ADC_Cmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

- Enables or disables the specified ADC peripheral.*

 - void [ADC_AnalogWatchdogCmd](#) (ADC_TypeDef *ADCx, uint32_t ADC_AnalogWatchdog)

Enables or disables the analog watchdog on single/all regular or injected channels.

 - void [ADC_AnalogWatchdogThresholdsConfig](#) (ADC_TypeDef *ADCx, uint16_t HighThreshold, uint16_t LowThreshold)

Configures the high and low thresholds of the analog watchdog.

 - void [ADC_AnalogWatchdogSingleChannelConfig](#) (ADC_TypeDef *ADCx, uint8_t ADC_Channel)

Configures the analog watchdog guarded single channel.

 - void [ADC_TempSensorVrefintCmd](#) (FunctionalState NewState)

Enables or disables the temperature sensor and Vrefint channels.

 - void [ADC_VBATCmd](#) (FunctionalState NewState)

Enables or disables the VBAT (Voltage Battery) channel.

 - void [ADC-RegularChannelConfig](#) (ADC_TypeDef *ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)

Configures for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

 - void [ADC_SoftwareStartConv](#) (ADC_TypeDef *ADCx)

Enables the selected ADC software start conversion of the regular channels.

 - FlagStatus [ADC_GetSoftwareStartConvStatus](#) (ADC_TypeDef *ADCx)

Gets the selected ADC Software start regular conversion Status.

 - void [ADC_EOCOnEachRegularChannelCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the EOC on each regular channel conversion.

 - void [ADC_ContinuousModeCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the ADC continuous conversion mode.

 - void [ADC_DiscModeChannelCountConfig](#) (ADC_TypeDef *ADCx, uint8_t Number)

Configures the discontinuous mode for the selected ADC regular group channel.

 - void [ADC_DiscModeCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the discontinuous mode on regular group channel for the specified ADC.

 - uint16_t [ADC_GetConversionValue](#) (ADC_TypeDef *ADCx)

Returns the last ADCx conversion result data for regular channel.

 - uint32_t [ADC_GetMultiModeConversionValue](#) (void)

Returns the last ADC1, ADC2 and ADC3 regular conversions results data in the selected multi mode.

 - void [ADC_DMAMCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the specified ADC DMA request.

 - void [ADC_DMARRequestAfterLastTransferCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)

Enables or disables the ADC DMA request after last transfer (Single-ADC mode)

 - void [ADC_MultiModeDMARRequestAfterLastTransferCmd](#) (FunctionalState NewState)

Enables or disables the ADC DMA request after last transfer in multi ADC mode

 - void [ADC_InjectedChannelConfig](#) (ADC_TypeDef *ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)

Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time.

 - void [ADC_InjectedSequencerLengthConfig](#) (ADC_TypeDef *ADCx, uint8_t Length)

Configures the sequencer length for injected channels.

 - void [ADC_SetInjectedOffset](#) (ADC_TypeDef *ADCx, uint8_t ADC_InjectedChannel, uint16_t Offset)

Set the injected channels conversion value offset.

 - void [ADC_ExternalTrigInjectedConvConfig](#) (ADC_TypeDef *ADCx, uint32_t ADC_ExternalTrigInjecConv)

Configures the ADCx external trigger for injected channels conversion.

 - void [ADC_ExternalTrigInjectedConvEdgeConfig](#) (ADC_TypeDef *ADCx, uint32_t ADC_ExternalTrigInjecConvEdge)

Configures the ADCx external trigger edge for injected channels conversion.

 - void [ADC_SoftwareStartInjectedConv](#) (ADC_TypeDef *ADCx)

- Enables the selected ADC software start conversion of the injected channels.*
- FlagStatus [ADC_GetSoftwareStartInjectedConvCmdStatus](#) (ADC_TypeDef *ADCx)
- Gets the selected ADC Software start injected conversion Status.*
- void [ADC_AutoInjectedConvCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
- Enables or disables the selected ADC automatic injected group conversion after regular one.*
- void [ADC_InjectedDiscModeCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
- Enables or disables the discontinuous mode for injected group channel for the specified ADC.*
- uint16_t [ADC_GetInjectedConversionValue](#) (ADC_TypeDef *ADCx, uint8_t ADC_InjectedChannel)
- Returns the ADC injected channel conversion result.*
- void [ADC_ITConfig](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT, FunctionalState NewState)
- Enables or disables the specified ADC interrupts.*
- FlagStatus [ADC_GetFlagStatus](#) (ADC_TypeDef *ADCx, uint8_t ADC_FLAG)
- Checks whether the specified ADC flag is set or not.*
- void [ADC_ClearFlag](#) (ADC_TypeDef *ADCx, uint8_t ADC_FLAG)
- Clears the ADCx's pending flags.*
- ITStatus [ADC_GetITStatus](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT)
- Checks whether the specified ADC interrupt has occurred or not.*
- void [ADC_ClearITPendingBit](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT)
- Clears the ADCx's interrupt pending bits.*

6.15.1 Detailed Description

This file provides firmware functions to manage the following functionalities of the Analog to Digital Converter (ADC) peripheral:

Author

MCD Application Team

Version

V1.0.0

Date

30-September-2011

- Initialization and Configuration (in addition to ADC multi mode selection)
- Analog Watchdog configuration
- Temperature Sensor & Vrefint (Voltage Reference internal) & VBAT management
- Regular Channels Configuration
- Regular Channels DMA Configuration
- Injected channels Configuration
- Interrupts and flags management

```

*
*  =====
*                                How to use this driver
*  =====
*
*  1. Enable the ADC interface clock using
*      RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADCx, ENABLE);
*
*  2. ADC pins configuration
*      - Enable the clock for the ADC GPIOs using the following function:
*          RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOx, ENABLE);
*      - Configure these ADC pins in analog mode using GPIO_Init();
*
*  3. Configure the ADC Prescaler, conversion resolution and data

```

```

*         alignment using the ADC_Init() function.
*     4. Activate the ADC peripheral using ADC_Cmd() function.
*
* Regular channels group configuration
* =====
*     - To configure the ADC regular channels group features, use
*       ADC_Init() and ADC_RegularChannelConfig() functions.
*     - To activate the continuous mode, use the ADC_continuousModeCmd()
*       function.
*     - To configurate and activate the Discontinuous mode, use the
*       ADC_DiscModeChannelCountConfig() and ADC_DiscModeCmd() functions.
*     - To read the ADC converted values, use the ADC_GetConversionValue()
*       function.
*
* Multi mode ADCs Regular channels configuration
* =====
*     - Refer to "Regular channels group configuration" description to
*       configure the ADC1, ADC2 and ADC3 regular channels.
*     - Select the Multi mode ADC regular channels features (dual or
*       triple mode) using ADC_CommonInit() function and configure
*       the DMA mode using ADC_MultiModeDMARequestAfterLastTransferCmd()
*       functions.
*     - Read the ADCs converted values using the
*       ADC_GetMultiModeConversionValue() function.
*
* DMA for Regular channels group features configuration
* =====
*     - To enable the DMA mode for regular channels group, use the
*       ADC_DMACmd() function.
*     - To enable the generation of DMA requests continuously at the end
*       of the last DMA transfer, use the ADC_DMARequestAfterLastTransferCmd()
*       function.
*
* Injected channels group configuration
* =====
*     - To configure the ADC Injected channels group features, use
*       ADC_InjectedChannelConfig() and ADC_InjectedSequencerLengthConfig()
*       functions.
*     - To activate the continuous mode, use the ADC_continuousModeCmd()
*       function.
*     - To activate the Injected Discontinuous mode, use the
*       ADC_InjectedDiscModeCmd() function.
*     - To activate the AutoInjected mode, use the ADC_AutoInjectedConvCmd()
*       function.
*     - To read the ADC converted values, use the ADC_GetInjectedConversionValue()
*       function.
*
*
*

```

Attention

THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

© COPYRIGHT 2011 STMicroelectronics

6.16 drivers/stm32f4xx_adc.h File Reference

This file contains all the functions prototypes for the ADC firmware library.

```
#include "stm32f4xx.h"
```

Data Structures

- struct [ADC_InitTypeDef](#)
ADC Init structure definition
- struct [ADC_CommonInitTypeDef](#)
ADC Common Init structure definition

Macros

- `#define IS_ADC_ALL_PERIPH(PERIPH)`
- `#define ADC_Mode_Independent ((uint32_t)0x00000000)`
- `#define ADC_DualMode_RegSimult_InjecSimult ((uint32_t)0x00000001)`
- `#define ADC_DualMode_RegSimult_AlterTrig ((uint32_t)0x00000002)`
- `#define ADC_DualMode_InjecSimult ((uint32_t)0x00000005)`
- `#define ADC_DualMode_RegSimult ((uint32_t)0x00000006)`
- `#define ADC_DualMode_Interl ((uint32_t)0x00000007)`
- `#define ADC_DualMode_AlterTrig ((uint32_t)0x00000009)`
- `#define ADC_TripleMode_RegSimult_InjecSimult ((uint32_t)0x00000011)`
- `#define ADC_TripleMode_RegSimult_AlterTrig ((uint32_t)0x00000012)`
- `#define ADC_TripleMode_InjecSimult ((uint32_t)0x00000015)`
- `#define ADC_TripleMode_RegSimult ((uint32_t)0x00000016)`
- `#define ADC_TripleMode_Interl ((uint32_t)0x00000017)`
- `#define ADC_TripleMode_AlterTrig ((uint32_t)0x00000019)`
- `#define IS_ADC_MODE(MODE)`
- `#define ADC_Prescaler_Div2 ((uint32_t)0x00000000)`
- `#define ADC_Prescaler_Div4 ((uint32_t)0x00010000)`
- `#define ADC_Prescaler_Div6 ((uint32_t)0x00020000)`
- `#define ADC_Prescaler_Div8 ((uint32_t)0x00030000)`
- `#define IS_ADC_PRESCALER(PRESCALER)`
- `#define ADC_DMAAccessMode_Disabled ((uint32_t)0x00000000) /* DMA mode disabled */`
- `#define ADC_DMAAccessMode_1 ((uint32_t)0x00004000) /* DMA mode 1 enabled (2 / 3 half-words one by one - 1 then 2 then 3)*/`
- `#define ADC_DMAAccessMode_2 ((uint32_t)0x00008000) /* DMA mode 2 enabled (2 / 3 half-words by pairs - 2&1 then 1&3 then 3&2)*/`
- `#define ADC_DMAAccessMode_3 ((uint32_t)0x0000C000) /* DMA mode 3 enabled (2 / 3 bytes by pairs - 2&1 then 1&3 then 3&2) */`
- `#define IS_ADC_DMA_ACCESS_MODE(MODE)`
- `#define ADC_TwoSamplingDelay_5Cycles ((uint32_t)0x00000000)`
- `#define ADC_TwoSamplingDelay_6Cycles ((uint32_t)0x00000100)`
- `#define ADC_TwoSamplingDelay_7Cycles ((uint32_t)0x00000200)`
- `#define ADC_TwoSamplingDelay_8Cycles ((uint32_t)0x00000300)`
- `#define ADC_TwoSamplingDelay_9Cycles ((uint32_t)0x00000400)`
- `#define ADC_TwoSamplingDelay_10Cycles ((uint32_t)0x00000500)`
- `#define ADC_TwoSamplingDelay_11Cycles ((uint32_t)0x00000600)`
- `#define ADC_TwoSamplingDelay_12Cycles ((uint32_t)0x00000700)`
- `#define ADC_TwoSamplingDelay_13Cycles ((uint32_t)0x00000800)`
- `#define ADC_TwoSamplingDelay_14Cycles ((uint32_t)0x00000900)`
- `#define ADC_TwoSamplingDelay_15Cycles ((uint32_t)0x00000A00)`
- `#define ADC_TwoSamplingDelay_16Cycles ((uint32_t)0x00000B00)`
- `#define ADC_TwoSamplingDelay_17Cycles ((uint32_t)0x00000C00)`
- `#define ADC_TwoSamplingDelay_18Cycles ((uint32_t)0x00000D00)`
- `#define ADC_TwoSamplingDelay_19Cycles ((uint32_t)0x00000E00)`
- `#define ADC_TwoSamplingDelay_20Cycles ((uint32_t)0x00000F00)`
- `#define IS_ADC_SAMPLING_DELAY(DELAY)`

- #define [ADC_Resolution_12b](#) ((uint32_t)0x00000000)
- #define [ADC_Resolution_10b](#) ((uint32_t)0x01000000)
- #define [ADC_Resolution_8b](#) ((uint32_t)0x02000000)
- #define [ADC_Resolution_6b](#) ((uint32_t)0x03000000)
- #define [IS_ADC_RESOLUTION](#)(RESOLUTION)
- #define [ADC_ExternalTrigConvEdge_None](#) ((uint32_t)0x00000000)
- #define [ADC_ExternalTrigConvEdge_Rising](#) ((uint32_t)0x10000000)
- #define [ADC_ExternalTrigConvEdge_Falling](#) ((uint32_t)0x20000000)
- #define [ADC_ExternalTrigConvEdge_RisingFalling](#) ((uint32_t)0x30000000)
- #define [IS_ADC_EXT_TRIG_EDGE](#)(EDGE)
- #define [ADC_ExternalTrigConv_T1_CC1](#) ((uint32_t)0x00000000)
- #define [ADC_ExternalTrigConv_T1_CC2](#) ((uint32_t)0x01000000)
- #define [ADC_ExternalTrigConv_T1_CC3](#) ((uint32_t)0x02000000)
- #define [ADC_ExternalTrigConv_T2_CC2](#) ((uint32_t)0x03000000)
- #define [ADC_ExternalTrigConv_T2_CC3](#) ((uint32_t)0x04000000)
- #define [ADC_ExternalTrigConv_T2_CC4](#) ((uint32_t)0x05000000)
- #define [ADC_ExternalTrigConv_T2_TRGO](#) ((uint32_t)0x06000000)
- #define [ADC_ExternalTrigConv_T3_CC1](#) ((uint32_t)0x07000000)
- #define [ADC_ExternalTrigConv_T3_TRGO](#) ((uint32_t)0x08000000)
- #define [ADC_ExternalTrigConv_T4_CC4](#) ((uint32_t)0x09000000)
- #define [ADC_ExternalTrigConv_T5_CC1](#) ((uint32_t)0x0A000000)
- #define [ADC_ExternalTrigConv_T5_CC2](#) ((uint32_t)0x0B000000)
- #define [ADC_ExternalTrigConv_T5_CC3](#) ((uint32_t)0x0C000000)
- #define [ADC_ExternalTrigConv_T8_CC1](#) ((uint32_t)0x0D000000)
- #define [ADC_ExternalTrigConv_T8_TRGO](#) ((uint32_t)0x0E000000)
- #define [ADC_ExternalTrigConv_Ext_IT11](#) ((uint32_t)0x0F000000)
- #define [IS_ADC_EXT_TRIG](#)(REGTRIG)
- #define [ADC_DataAlign_Right](#) ((uint32_t)0x00000000)
- #define [ADC_DataAlign_Left](#) ((uint32_t)0x00000800)
- #define [IS_ADC_DATA_ALIGN](#)(ALIGN)
- #define [ADC_Channel_0](#) ((uint8_t)0x00)
- #define [ADC_Channel_1](#) ((uint8_t)0x01)
- #define [ADC_Channel_2](#) ((uint8_t)0x02)
- #define [ADC_Channel_3](#) ((uint8_t)0x03)
- #define [ADC_Channel_4](#) ((uint8_t)0x04)
- #define [ADC_Channel_5](#) ((uint8_t)0x05)
- #define [ADC_Channel_6](#) ((uint8_t)0x06)
- #define [ADC_Channel_7](#) ((uint8_t)0x07)
- #define [ADC_Channel_8](#) ((uint8_t)0x08)
- #define [ADC_Channel_9](#) ((uint8_t)0x09)
- #define [ADC_Channel_10](#) ((uint8_t)0x0A)
- #define [ADC_Channel_11](#) ((uint8_t)0x0B)
- #define [ADC_Channel_12](#) ((uint8_t)0x0C)
- #define [ADC_Channel_13](#) ((uint8_t)0x0D)
- #define [ADC_Channel_14](#) ((uint8_t)0x0E)
- #define [ADC_Channel_15](#) ((uint8_t)0x0F)
- #define [ADC_Channel_16](#) ((uint8_t)0x10)
- #define [ADC_Channel_17](#) ((uint8_t)0x11)
- #define [ADC_Channel_18](#) ((uint8_t)0x12)
- #define [ADC_Channel_TempSensor](#) ((uint8_t)[ADC_Channel_16](#))
- #define [ADC_Channel_Vrefint](#) ((uint8_t)[ADC_Channel_17](#))
- #define [ADC_Channel_Vbat](#) ((uint8_t)[ADC_Channel_18](#))
- #define [IS_ADC_CHANNEL](#)(CHANNEL)
- #define [ADC_SampleTime_3Cycles](#) ((uint8_t)0x00)
- #define [ADC_SampleTime_15Cycles](#) ((uint8_t)0x01)

- #define `ADC_SampleTime_28Cycles` ((uint8_t)0x02)
- #define `ADC_SampleTime_56Cycles` ((uint8_t)0x03)
- #define `ADC_SampleTime_84Cycles` ((uint8_t)0x04)
- #define `ADC_SampleTime_112Cycles` ((uint8_t)0x05)
- #define `ADC_SampleTime_144Cycles` ((uint8_t)0x06)
- #define `ADC_SampleTime_480Cycles` ((uint8_t)0x07)
- #define `IS_ADC_SAMPLE_TIME`(TIME)
- #define `ADC_ExternalTrigInjecConvEdge_None` ((uint32_t)0x00000000)
- #define `ADC_ExternalTrigInjecConvEdge_Rising` ((uint32_t)0x00100000)
- #define `ADC_ExternalTrigInjecConvEdge_Falling` ((uint32_t)0x00200000)
- #define `ADC_ExternalTrigInjecConvEdge_RisingFalling` ((uint32_t)0x00300000)
- #define `IS_ADC_EXT_INJEC_TRIG_EDGE`(EDGE)
- #define `ADC_ExternalTrigInjecConv_T1_CC4` ((uint32_t)0x00000000)
- #define `ADC_ExternalTrigInjecConv_T1_TRGO` ((uint32_t)0x00010000)
- #define `ADC_ExternalTrigInjecConv_T2_CC1` ((uint32_t)0x00020000)
- #define `ADC_ExternalTrigInjecConv_T2_TRGO` ((uint32_t)0x00030000)
- #define `ADC_ExternalTrigInjecConv_T3_CC2` ((uint32_t)0x00040000)
- #define `ADC_ExternalTrigInjecConv_T3_CC4` ((uint32_t)0x00050000)
- #define `ADC_ExternalTrigInjecConv_T4_CC1` ((uint32_t)0x00060000)
- #define `ADC_ExternalTrigInjecConv_T4_CC2` ((uint32_t)0x00070000)
- #define `ADC_ExternalTrigInjecConv_T4_CC3` ((uint32_t)0x00080000)
- #define `ADC_ExternalTrigInjecConv_T4_TRGO` ((uint32_t)0x00090000)
- #define `ADC_ExternalTrigInjecConv_T5_CC4` ((uint32_t)0x000A0000)
- #define `ADC_ExternalTrigInjecConv_T5_TRGO` ((uint32_t)0x000B0000)
- #define `ADC_ExternalTrigInjecConv_T8_CC2` ((uint32_t)0x000C0000)
- #define `ADC_ExternalTrigInjecConv_T8_CC3` ((uint32_t)0x000D0000)
- #define `ADC_ExternalTrigInjecConv_T8_CC4` ((uint32_t)0x000E0000)
- #define `ADC_ExternalTrigInjecConv_Ext_IT15` ((uint32_t)0x000F0000)
- #define `IS_ADC_EXT_INJEC_TRIG`(INJTRIG)
- #define `ADC_InjectedChannel_1` ((uint8_t)0x14)
- #define `ADC_InjectedChannel_2` ((uint8_t)0x18)
- #define `ADC_InjectedChannel_3` ((uint8_t)0x1C)
- #define `ADC_InjectedChannel_4` ((uint8_t)0x20)
- #define `IS_ADC_INJECTED_CHANNEL`(CHANNEL)
- #define `ADC_AnalogWatchdog_SingleRegEnable` ((uint32_t)0x00800200)
- #define `ADC_AnalogWatchdog_SingleInjecEnable` ((uint32_t)0x00400200)
- #define `ADC_AnalogWatchdog_SingleRegOrInjecEnable` ((uint32_t)0x00C00200)
- #define `ADC_AnalogWatchdog_AllRegEnable` ((uint32_t)0x00800000)
- #define `ADC_AnalogWatchdog_AllInjecEnable` ((uint32_t)0x00400000)
- #define `ADC_AnalogWatchdog_AllRegAllInjecEnable` ((uint32_t)0x00C00000)
- #define `ADC_AnalogWatchdog_None` ((uint32_t)0x00000000)
- #define `IS_ADC_ANALOG_WATCHDOG`(WATCHDOG)
- #define `ADC_IT_EOC` ((uint16_t)0x0205)
- #define `ADC_IT_AWD` ((uint16_t)0x0106)
- #define `ADC_IT_JEOC` ((uint16_t)0x0407)
- #define `ADC_IT_OVR` ((uint16_t)0x201A)
- #define `IS_ADC_IT`(IT)
- #define `ADC_FLAG_AWD` ((uint8_t)0x01)
- #define `ADC_FLAG_EOC` ((uint8_t)0x02)
- #define `ADC_FLAG_JEOC` ((uint8_t)0x04)
- #define `ADC_FLAG_JSTRT` ((uint8_t)0x08)
- #define `ADC_FLAG_STRT` ((uint8_t)0x10)
- #define `ADC_FLAG_OVR` ((uint8_t)0x20)
- #define `IS_ADC_CLEAR_FLAG`(FLAG) (((FLAG) & (uint8_t)0xC0) == 0x00) && ((FLAG) != 0x00)
- #define `IS_ADC_GET_FLAG`(FLAG)

- `#define IS_ADC_THRESHOLD(THRESHOLD) ((THRESHOLD) <= 0xFFF)`
- `#define IS_ADC_OFFSET(OFFSET) ((OFFSET) <= 0xFFF)`
- `#define IS_ADC_INJECTED_LENGTH(LENGTH) (((LENGTH) >= 0x1) && ((LENGTH) <= 0x4))`
- `#define IS_ADC_INJECTED_RANK(RANK) (((RANK) >= 0x1) && ((RANK) <= 0x4))`
- `#define IS_ADC_REGULAR_LENGTH(LENGTH) (((LENGTH) >= 0x1) && ((LENGTH) <= 0x10))`
- `#define IS_ADC_REGULAR_RANK(RANK) (((RANK) >= 0x1) && ((RANK) <= 0x10))`
- `#define IS_ADC_REGULAR_DISC_NUMBER(NUMBER) (((NUMBER) >= 0x1) && ((NUMBER) <= 0x8))`

Functions

- void `ADC_DeInit` (void)
Deinitializes all ADCs peripherals registers to their default reset values.
- void `ADC_Init` (ADC_TypeDef *ADCx, ADC_InitTypeDef *ADC_InitStruct)
Initializes the ADCx peripheral according to the specified parameters in the ADC_InitStruct.
- void `ADC_StructInit` (ADC_InitTypeDef *ADC_InitStruct)
Fills each ADC_InitStruct member with its default value.
- void `ADC_CommonInit` (ADC_CommonInitTypeDef *ADC_CommonInitStruct)
Initializes the ADCs peripherals according to the specified parameters in the ADC_CommonInitStruct.
- void `ADC_CommonStructInit` (ADC_CommonInitTypeDef *ADC_CommonInitStruct)
Fills each ADC_CommonInitStruct member with its default value.
- void `ADC_Cmd` (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the specified ADC peripheral.
- void `ADC_AnalogWatchdogCmd` (ADC_TypeDef *ADCx, uint32_t ADC_AnalogWatchdog)
Enables or disables the analog watchdog on single/all regular or injected channels.
- void `ADC_AnalogWatchdogThresholdsConfig` (ADC_TypeDef *ADCx, uint16_t HighThreshold, uint16_t LowThreshold)
Configures the high and low thresholds of the analog watchdog.
- void `ADC_AnalogWatchdogSingleChannelConfig` (ADC_TypeDef *ADCx, uint8_t ADC_Channel)
Configures the analog watchdog guarded single channel.
- void `ADC_TempSensorVrefintCmd` (FunctionalState NewState)
Enables or disables the temperature sensor and Vrefint channels.
- void `ADC_VBATCmd` (FunctionalState NewState)
Enables or disables the VBAT (Voltage Battery) channel.
- void `ADC-RegularChannelConfig` (ADC_TypeDef *ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)
Configures for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
- void `ADC_SoftwareStartConv` (ADC_TypeDef *ADCx)
Enables the selected ADC software start conversion of the regular channels.
- FlagStatus `ADC_GetSoftwareStartConvStatus` (ADC_TypeDef *ADCx)
Gets the selected ADC Software start regular conversion Status.
- void `ADC_EOCOnEachRegularChannelCmd` (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the EOC on each regular channel conversion.
- void `ADC_ContinuousModeCmd` (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the ADC continuous conversion mode.
- void `ADC_DiscModeChannelCountConfig` (ADC_TypeDef *ADCx, uint8_t Number)
Configures the discontinuous mode for the selected ADC regular group channel.
- void `ADC_DiscModeCmd` (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the discontinuous mode on regular group channel for the specified ADC.
- uint16_t `ADC_GetConversionValue` (ADC_TypeDef *ADCx)
Returns the last ADCx conversion result data for regular channel.
- uint32_t `ADC_GetMultiModeConversionValue` (void)
Returns the last ADC1, ADC2 and ADC3 regular conversions results data in the selected multi mode.

- void [ADC_DMAMCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the specified ADC DMA request.
- void [ADC_DMAResultRequestAfterLastTransferCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the ADC DMA request after last transfer (Single-ADC mode)
- void [ADC_MultiModeDMAResultRequestAfterLastTransferCmd](#) (FunctionalState NewState)
Enables or disables the ADC DMA request after last transfer in multi ADC mode
- void [ADC_InjectedChannelConfig](#) (ADC_TypeDef *ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)
Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time.
- void [ADC_InjectedSequencerLengthConfig](#) (ADC_TypeDef *ADCx, uint8_t Length)
Configures the sequencer length for injected channels.
- void [ADC_SetInjectedOffset](#) (ADC_TypeDef *ADCx, uint8_t ADC_InjectedChannel, uint16_t Offset)
Set the injected channels conversion value offset.
- void [ADC_ExternalTrigInjectedConvConfig](#) (ADC_TypeDef *ADCx, uint32_t ADC_ExternalTrigInjecConv)
Configures the ADCx external trigger for injected channels conversion.
- void [ADC_ExternalTrigInjectedConvEdgeConfig](#) (ADC_TypeDef *ADCx, uint32_t ADC_ExternalTrigInjecConvEdge)
Configures the ADCx external trigger edge for injected channels conversion.
- void [ADC_SoftwareStartInjectedConv](#) (ADC_TypeDef *ADCx)
Enables the selected ADC software start conversion of the injected channels.
- FlagStatus [ADC_GetSoftwareStartInjectedConvCmdStatus](#) (ADC_TypeDef *ADCx)
Gets the selected ADC Software start injected conversion Status.
- void [ADC_AutoInjectedConvCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the selected ADC automatic injected group conversion after regular one.
- void [ADC_InjectedDiscModeCmd](#) (ADC_TypeDef *ADCx, FunctionalState NewState)
Enables or disables the discontinuous mode for injected group channel for the specified ADC.
- uint16_t [ADC_GetInjectedConversionValue](#) (ADC_TypeDef *ADCx, uint8_t ADC_InjectedChannel)
Returns the ADC injected channel conversion result.
- void [ADC_ITConfig](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT, FunctionalState NewState)
Enables or disables the specified ADC interrupts.
- FlagStatus [ADC_GetFlagStatus](#) (ADC_TypeDef *ADCx, uint8_t ADC_FLAG)
Checks whether the specified ADC flag is set or not.
- void [ADC_ClearFlag](#) (ADC_TypeDef *ADCx, uint8_t ADC_FLAG)
Clears the ADCx's pending flags.
- ITStatus [ADC_GetITStatus](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT)
Checks whether the specified ADC interrupt has occurred or not.
- void [ADC_ClearITPendingBit](#) (ADC_TypeDef *ADCx, uint16_t ADC_IT)
Clears the ADCx's interrupt pending bits.

6.16.1 Detailed Description

This file contains all the functions prototypes for the ADC firmware library.

Author

MCD Application Team

Version

V1.0.0

Date

30-September-2011

Attention

THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

© COPYRIGHT 2011 STMicroelectronics

6.17 stm32f4xx_adc.h

[Go to the documentation of this file.](#)

```

00001
00023 /* Define to prevent recursive inclusion -----*/
00024 #ifndef __STM32F4xx_ADC_H
00025 #define __STM32F4xx_ADC_H
00026
00027 #ifdef __cplusplus
00028     extern "C" {
00029 #endif
00030
00031 /* Includes -----*/
00032 #include "stm32f4xx.h"
00033
00042 /* Exported types -----*/
00043
00047 typedef struct
00048 {
00049     uint32_t ADC_Resolution;
00051     FunctionalState ADC_ScanConvMode;
00055     FunctionalState ADC_ContinuousConvMode;
00058     uint32_t ADC_ExternalTrigConvEdge;
00062     uint32_t ADC_ExternalTrigConv;
00066     uint32_t ADC_DataAlign;
00069     uint8_t ADC_NbrOfConversion;
00073 }ADC_InitTypeDef;
00074
00078 typedef struct
00079 {
00080     uint32_t ADC_Mode;
00083     uint32_t ADC_Prescaler;
00086     uint32_t ADC_DMAAccessMode;
00090     uint32_t ADC_TwoSamplingDelay;
00094 }ADC_CommonInitTypeDef;
00095
00096
00097 /* Exported constants -----*/
00098
00102 #define IS_ADC_ALL_PERIPH(PERIPH) (((PERIPH) == ADC1) || \
00103                                     ((PERIPH) == ADC2) || \
00104                                     ((PERIPH) == ADC3))
00105
00109 #define ADC_Mode_Independent ((uint32_t)0x00000000)
00110 #define ADC_DualMode_RegSimult_InjecSimult ((uint32_t)0x00000001)
00111 #define ADC_DualMode_RegSimult_AlterTrig ((uint32_t)0x00000002)
00112 #define ADC_DualMode_InjecSimult ((uint32_t)0x00000005)
00113 #define ADC_DualMode_RegSimult ((uint32_t)0x00000006)
00114 #define ADC_DualMode_Interl ((uint32_t)0x00000007)
00115 #define ADC_DualMode_AlterTrig ((uint32_t)0x00000009)
00116 #define ADC_TripleMode_RegSimult_InjecSimult ((uint32_t)0x00000011)
00117 #define ADC_TripleMode_RegSimult_AlterTrig ((uint32_t)0x00000012)
00118 #define ADC_TripleMode_InjecSimult ((uint32_t)0x00000015)
00119 #define ADC_TripleMode_RegSimult ((uint32_t)0x00000016)
00120 #define ADC_TripleMode_Interl ((uint32_t)0x00000017)
00121 #define ADC_TripleMode_AlterTrig ((uint32_t)0x00000019)
00122 #define IS_ADC_MODE(MODE) (((MODE) == ADC_Mode_Independent) || \
00123                             ((MODE) == ADC_DualMode_RegSimult_InjecSimult) || \
00124                             ((MODE) == ADC_DualMode_RegSimult_AlterTrig) || \

```

```

00125             ((MODE) == ADC_DualMode_InjecSimult) || \
00126             ((MODE) == ADC_DualMode_RegSimult) || \
00127             ((MODE) == ADC_DualMode_Interl) || \
00128             ((MODE) == ADC_DualMode_AlterTrig) || \
00129             ((MODE) == ADC_TripleMode_RegSimult_InjecSimult) || \
00130             ((MODE) == ADC_TripleMode_RegSimult_AlterTrig) || \
00131             ((MODE) == ADC_TripleMode_InjecSimult) || \
00132             ((MODE) == ADC_TripleMode_RegSimult) || \
00133             ((MODE) == ADC_TripleMode_Interl) || \
00134             ((MODE) == ADC_TripleMode_AlterTrig))
00143 #define ADC_Prescaler_Div2 ((uint32_t)0x00000000)
00144 #define ADC_Prescaler_Div4 ((uint32_t)0x00010000)
00145 #define ADC_Prescaler_Div6 ((uint32_t)0x00020000)
00146 #define ADC_Prescaler_Div8 ((uint32_t)0x00030000)
00147 #define IS_ADC_PRESCALER(PRESCALER) (((PRESCALER) == ADC_Prescaler_Div2) || \
00148                                     ((PRESCALER) == ADC_Prescaler_Div4) || \
00149                                     ((PRESCALER) == ADC_Prescaler_Div6) || \
00150                                     ((PRESCALER) == ADC_Prescaler_Div8))
00159 #define ADC_DMAAccessMode_Disabled ((uint32_t)0x00000000) /* DMA mode disabled */
00160 #define ADC_DMAAccessMode_1 ((uint32_t)0x00004000) /* DMA mode 1 enabled (2 / 3
half-words one by one - 1 then 2 then 3) */
00161 #define ADC_DMAAccessMode_2 ((uint32_t)0x00008000) /* DMA mode 2 enabled (2 / 3
half-words by pairs - 2&1 then 1&3 then 3&2) */
00162 #define ADC_DMAAccessMode_3 ((uint32_t)0x0000C000) /* DMA mode 3 enabled (2 / 3 bytes
by pairs - 2&1 then 1&3 then 3&2) */
00163 #define IS_ADC_DMA_ACCESS_MODE(MODE) (((MODE) == ADC_DMAAccessMode_Disabled) || \
00164                                     ((MODE) == ADC_DMAAccessMode_1) || \
00165                                     ((MODE) == ADC_DMAAccessMode_2) || \
00166                                     ((MODE) == ADC_DMAAccessMode_3))
00167
00176 #define ADC_TwoSamplingDelay_5Cycles ((uint32_t)0x00000000)
00177 #define ADC_TwoSamplingDelay_6Cycles ((uint32_t)0x00000100)
00178 #define ADC_TwoSamplingDelay_7Cycles ((uint32_t)0x00000200)
00179 #define ADC_TwoSamplingDelay_8Cycles ((uint32_t)0x00000300)
00180 #define ADC_TwoSamplingDelay_9Cycles ((uint32_t)0x00000400)
00181 #define ADC_TwoSamplingDelay_10Cycles ((uint32_t)0x00000500)
00182 #define ADC_TwoSamplingDelay_11Cycles ((uint32_t)0x00000600)
00183 #define ADC_TwoSamplingDelay_12Cycles ((uint32_t)0x00000700)
00184 #define ADC_TwoSamplingDelay_13Cycles ((uint32_t)0x00000800)
00185 #define ADC_TwoSamplingDelay_14Cycles ((uint32_t)0x00000900)
00186 #define ADC_TwoSamplingDelay_15Cycles ((uint32_t)0x00000A00)
00187 #define ADC_TwoSamplingDelay_16Cycles ((uint32_t)0x00000B00)
00188 #define ADC_TwoSamplingDelay_17Cycles ((uint32_t)0x00000C00)
00189 #define ADC_TwoSamplingDelay_18Cycles ((uint32_t)0x00000D00)
00190 #define ADC_TwoSamplingDelay_19Cycles ((uint32_t)0x00000E00)
00191 #define ADC_TwoSamplingDelay_20Cycles ((uint32_t)0x00000F00)
00192 #define IS_ADC_SAMPLING_DELAY(DELAY) (((DELAY) == ADC_TwoSamplingDelay_5Cycles) || \
00193                                     ((DELAY) == ADC_TwoSamplingDelay_6Cycles) || \
00194                                     ((DELAY) == ADC_TwoSamplingDelay_7Cycles) || \
00195                                     ((DELAY) == ADC_TwoSamplingDelay_8Cycles) || \
00196                                     ((DELAY) == ADC_TwoSamplingDelay_9Cycles) || \
00197                                     ((DELAY) == ADC_TwoSamplingDelay_10Cycles) || \
00198                                     ((DELAY) == ADC_TwoSamplingDelay_11Cycles) || \
00199                                     ((DELAY) == ADC_TwoSamplingDelay_12Cycles) || \
00200                                     ((DELAY) == ADC_TwoSamplingDelay_13Cycles) || \
00201                                     ((DELAY) == ADC_TwoSamplingDelay_14Cycles) || \
00202                                     ((DELAY) == ADC_TwoSamplingDelay_15Cycles) || \
00203                                     ((DELAY) == ADC_TwoSamplingDelay_16Cycles) || \
00204                                     ((DELAY) == ADC_TwoSamplingDelay_17Cycles) || \
00205                                     ((DELAY) == ADC_TwoSamplingDelay_18Cycles) || \
00206                                     ((DELAY) == ADC_TwoSamplingDelay_19Cycles) || \
00207                                     ((DELAY) == ADC_TwoSamplingDelay_20Cycles))
00208
00217 #define ADC_Resolution_12b ((uint32_t)0x00000000)
00218 #define ADC_Resolution_10b ((uint32_t)0x01000000)
00219 #define ADC_Resolution_8b ((uint32_t)0x02000000)
00220 #define ADC_Resolution_6b ((uint32_t)0x03000000)
00221 #define IS_ADC_RESOLUTION(RESOLUTION) (((RESOLUTION) == ADC_Resolution_12b) || \
00222                                     ((RESOLUTION) == ADC_Resolution_10b) || \
00223                                     ((RESOLUTION) == ADC_Resolution_8b) || \
00224                                     ((RESOLUTION) == ADC_Resolution_6b))
00225
00234 #define ADC_ExternalTrigConvEdge_None ((uint32_t)0x00000000)
00235 #define ADC_ExternalTrigConvEdge_Rising ((uint32_t)0x10000000)
00236 #define ADC_ExternalTrigConvEdge_Falling ((uint32_t)0x20000000)
00237 #define ADC_ExternalTrigConvEdge_RisingFalling ((uint32_t)0x30000000)
00238 #define IS_ADC_EXT_TRIG_EDGE(EDGE) ((EDGE) == ADC_ExternalTrigConvEdge_None) || \
00239                                     ((EDGE) == ADC_ExternalTrigConvEdge_Rising) || \
00240                                     ((EDGE) == ADC_ExternalTrigConvEdge_Falling) || \
00241                                     ((EDGE) == ADC_ExternalTrigConvEdge_RisingFalling))
00250 #define ADC_ExternalTrigConv_T1_CC1 ((uint32_t)0x00000000)
00251 #define ADC_ExternalTrigConv_T1_CC2 ((uint32_t)0x01000000)
00252 #define ADC_ExternalTrigConv_T1_CC3 ((uint32_t)0x02000000)
00253 #define ADC_ExternalTrigConv_T2_CC2 ((uint32_t)0x03000000)
00254 #define ADC_ExternalTrigConv_T2_CC3 ((uint32_t)0x04000000)
00255 #define ADC_ExternalTrigConv_T2_CC4 ((uint32_t)0x05000000)
00256 #define ADC_ExternalTrigConv_T2_TRGO ((uint32_t)0x06000000)

```



```

00257 #define ADC_ExternalTrigConv_T3_CC1 ((uint32_t)0x07000000)
00258 #define ADC_ExternalTrigConv_T3_TRGO ((uint32_t)0x08000000)
00259 #define ADC_ExternalTrigConv_T4_CC4 ((uint32_t)0x09000000)
00260 #define ADC_ExternalTrigConv_T5_CC1 ((uint32_t)0x0A000000)
00261 #define ADC_ExternalTrigConv_T5_CC2 ((uint32_t)0x0B000000)
00262 #define ADC_ExternalTrigConv_T5_CC3 ((uint32_t)0x0C000000)
00263 #define ADC_ExternalTrigConv_T8_CC1 ((uint32_t)0x0D000000)
00264 #define ADC_ExternalTrigConv_T8_TRGO ((uint32_t)0x0E000000)
00265 #define ADC_ExternalTrigConv_Ext_IT11 ((uint32_t)0x0F000000)
00266 #define IS_ADC_EXT_TRIG(REGTRIG) (((REGTRIG) == ADC_ExternalTrigConv_T1_CC1) || \
00267 ((REGTRIG) == ADC_ExternalTrigConv_T1_CC2) || \
00268 ((REGTRIG) == ADC_ExternalTrigConv_T1_CC3) || \
00269 ((REGTRIG) == ADC_ExternalTrigConv_T2_CC2) || \
00270 ((REGTRIG) == ADC_ExternalTrigConv_T2_CC3) || \
00271 ((REGTRIG) == ADC_ExternalTrigConv_T2_CC4) || \
00272 ((REGTRIG) == ADC_ExternalTrigConv_T2_TRGO) || \
00273 ((REGTRIG) == ADC_ExternalTrigConv_T3_CC1) || \
00274 ((REGTRIG) == ADC_ExternalTrigConv_T3_TRGO) || \
00275 ((REGTRIG) == ADC_ExternalTrigConv_T4_CC4) || \
00276 ((REGTRIG) == ADC_ExternalTrigConv_T5_CC1) || \
00277 ((REGTRIG) == ADC_ExternalTrigConv_T5_CC2) || \
00278 ((REGTRIG) == ADC_ExternalTrigConv_T5_CC3) || \
00279 ((REGTRIG) == ADC_ExternalTrigConv_T8_CC1) || \
00280 ((REGTRIG) == ADC_ExternalTrigConv_T8_TRGO) || \
00281 ((REGTRIG) == ADC_ExternalTrigConv_Ext_IT11))
00290 #define ADC_DataAlign_Right ((uint32_t)0x00000000)
00291 #define ADC_DataAlign_Left ((uint32_t)0x00000800)
00292 #define IS_ADC_DATA_ALIGN(ALIGN) (((ALIGN) == ADC_DataAlign_Right) || \
00293 ((ALIGN) == ADC_DataAlign_Left))
00302 #define ADC_Channel_0 ((uint8_t)0x00)
00303 #define ADC_Channel_1 ((uint8_t)0x01)
00304 #define ADC_Channel_2 ((uint8_t)0x02)
00305 #define ADC_Channel_3 ((uint8_t)0x03)
00306 #define ADC_Channel_4 ((uint8_t)0x04)
00307 #define ADC_Channel_5 ((uint8_t)0x05)
00308 #define ADC_Channel_6 ((uint8_t)0x06)
00309 #define ADC_Channel_7 ((uint8_t)0x07)
00310 #define ADC_Channel_8 ((uint8_t)0x08)
00311 #define ADC_Channel_9 ((uint8_t)0x09)
00312 #define ADC_Channel_10 ((uint8_t)0x0A)
00313 #define ADC_Channel_11 ((uint8_t)0x0B)
00314 #define ADC_Channel_12 ((uint8_t)0x0C)
00315 #define ADC_Channel_13 ((uint8_t)0x0D)
00316 #define ADC_Channel_14 ((uint8_t)0x0E)
00317 #define ADC_Channel_15 ((uint8_t)0x0F)
00318 #define ADC_Channel_16 ((uint8_t)0x10)
00319 #define ADC_Channel_17 ((uint8_t)0x11)
00320 #define ADC_Channel_18 ((uint8_t)0x12)
00321
00322 #define ADC_Channel_TempSensor ((uint8_t)ADC_Channel_16)
00323 #define ADC_Channel_Vrefint ((uint8_t)ADC_Channel_17)
00324 #define ADC_Channel_Vbat ((uint8_t)ADC_Channel_18)
00325
00326 #define IS_ADC_CHANNEL(CHANNEL) (((CHANNEL) == ADC_Channel_0) || \
00327 ((CHANNEL) == ADC_Channel_1) || \
00328 ((CHANNEL) == ADC_Channel_2) || \
00329 ((CHANNEL) == ADC_Channel_3) || \
00330 ((CHANNEL) == ADC_Channel_4) || \
00331 ((CHANNEL) == ADC_Channel_5) || \
00332 ((CHANNEL) == ADC_Channel_6) || \
00333 ((CHANNEL) == ADC_Channel_7) || \
00334 ((CHANNEL) == ADC_Channel_8) || \
00335 ((CHANNEL) == ADC_Channel_9) || \
00336 ((CHANNEL) == ADC_Channel_10) || \
00337 ((CHANNEL) == ADC_Channel_11) || \
00338 ((CHANNEL) == ADC_Channel_12) || \
00339 ((CHANNEL) == ADC_Channel_13) || \
00340 ((CHANNEL) == ADC_Channel_14) || \
00341 ((CHANNEL) == ADC_Channel_15) || \
00342 ((CHANNEL) == ADC_Channel_16) || \
00343 ((CHANNEL) == ADC_Channel_17) || \
00344 ((CHANNEL) == ADC_Channel_18))
00353 #define ADC_SampleTime_3Cycles ((uint8_t)0x00)
00354 #define ADC_SampleTime_15Cycles ((uint8_t)0x01)
00355 #define ADC_SampleTime_28Cycles ((uint8_t)0x02)
00356 #define ADC_SampleTime_56Cycles ((uint8_t)0x03)
00357 #define ADC_SampleTime_84Cycles ((uint8_t)0x04)
00358 #define ADC_SampleTime_112Cycles ((uint8_t)0x05)
00359 #define ADC_SampleTime_144Cycles ((uint8_t)0x06)
00360 #define ADC_SampleTime_480Cycles ((uint8_t)0x07)
00361 #define IS_ADC_SAMPLE_TIME(TIME) (((TIME) == ADC_SampleTime_3Cycles) || \
00362 ((TIME) == ADC_SampleTime_15Cycles) || \
00363 ((TIME) == ADC_SampleTime_28Cycles) || \
00364 ((TIME) == ADC_SampleTime_56Cycles) || \
00365 ((TIME) == ADC_SampleTime_84Cycles) || \
00366 ((TIME) == ADC_SampleTime_112Cycles) || \
00367 ((TIME) == ADC_SampleTime_144Cycles) || \

```

```

00368 ((TIME) == ADC_SampleTime_480Cycles))
00377 #define ADC_ExternalTrigInjecConvEdge_None ((uint32_t)0x00000000)
00378 #define ADC_ExternalTrigInjecConvEdge_Rising ((uint32_t)0x00100000)
00379 #define ADC_ExternalTrigInjecConvEdge_Falling ((uint32_t)0x00200000)
00380 #define ADC_ExternalTrigInjecConvEdge_RisingFalling ((uint32_t)0x00300000)
00381 #define IS_ADC_EXT_INJEC_TRIG_EDGE(EDGE) (((EDGE) == ADC_ExternalTrigInjecConvEdge_None) || \
00382 ((EDGE) == ADC_ExternalTrigInjecConvEdge_Rising) || \
00383 ((EDGE) == ADC_ExternalTrigInjecConvEdge_Falling) || \
00384 ((EDGE) == ADC_ExternalTrigInjecConvEdge_RisingFalling))
00385
00394 #define ADC_ExternalTrigInjecConv_T1_CC4 ((uint32_t)0x00000000)
00395 #define ADC_ExternalTrigInjecConv_T1_TRGO ((uint32_t)0x00010000)
00396 #define ADC_ExternalTrigInjecConv_T2_CC1 ((uint32_t)0x00020000)
00397 #define ADC_ExternalTrigInjecConv_T2_TRGO ((uint32_t)0x00030000)
00398 #define ADC_ExternalTrigInjecConv_T3_CC2 ((uint32_t)0x00040000)
00399 #define ADC_ExternalTrigInjecConv_T3_CC4 ((uint32_t)0x00050000)
00400 #define ADC_ExternalTrigInjecConv_T4_CC1 ((uint32_t)0x00060000)
00401 #define ADC_ExternalTrigInjecConv_T4_CC2 ((uint32_t)0x00070000)
00402 #define ADC_ExternalTrigInjecConv_T4_CC3 ((uint32_t)0x00080000)
00403 #define ADC_ExternalTrigInjecConv_T4_TRGO ((uint32_t)0x00090000)
00404 #define ADC_ExternalTrigInjecConv_T5_CC4 ((uint32_t)0x000A0000)
00405 #define ADC_ExternalTrigInjecConv_T5_TRGO ((uint32_t)0x000B0000)
00406 #define ADC_ExternalTrigInjecConv_T8_CC2 ((uint32_t)0x000C0000)
00407 #define ADC_ExternalTrigInjecConv_T8_CC3 ((uint32_t)0x000D0000)
00408 #define ADC_ExternalTrigInjecConv_T8_CC4 ((uint32_t)0x000E0000)
00409 #define ADC_ExternalTrigInjecConv_Ext_IT15 ((uint32_t)0x000F0000)
00410 #define IS_ADC_EXT_INJEC_TRIG(INJTRIG) (((INJTRIG) == ADC_ExternalTrigInjecConv_T1_CC4) || \
00411 ((INJTRIG) == ADC_ExternalTrigInjecConv_T1_TRGO) || \
00412 ((INJTRIG) == ADC_ExternalTrigInjecConv_T2_CC1) || \
00413 ((INJTRIG) == ADC_ExternalTrigInjecConv_T2_TRGO) || \
00414 ((INJTRIG) == ADC_ExternalTrigInjecConv_T3_CC2) || \
00415 ((INJTRIG) == ADC_ExternalTrigInjecConv_T3_CC4) || \
00416 ((INJTRIG) == ADC_ExternalTrigInjecConv_T4_CC1) || \
00417 ((INJTRIG) == ADC_ExternalTrigInjecConv_T4_CC2) || \
00418 ((INJTRIG) == ADC_ExternalTrigInjecConv_T4_CC3) || \
00419 ((INJTRIG) == ADC_ExternalTrigInjecConv_T4_TRGO) || \
00420 ((INJTRIG) == ADC_ExternalTrigInjecConv_T5_CC4) || \
00421 ((INJTRIG) == ADC_ExternalTrigInjecConv_T5_TRGO) || \
00422 ((INJTRIG) == ADC_ExternalTrigInjecConv_T8_CC2) || \
00423 ((INJTRIG) == ADC_ExternalTrigInjecConv_T8_CC3) || \
00424 ((INJTRIG) == ADC_ExternalTrigInjecConv_T8_CC4) || \
00425 ((INJTRIG) == ADC_ExternalTrigInjecConv_Ext_IT15))
00434 #define ADC_InjectedChannel_1 ((uint8_t)0x14)
00435 #define ADC_InjectedChannel_2 ((uint8_t)0x18)
00436 #define ADC_InjectedChannel_3 ((uint8_t)0x1C)
00437 #define ADC_InjectedChannel_4 ((uint8_t)0x20)
00438 #define IS_ADC_INJECTED_CHANNEL(CHANNEL) (((CHANNEL) == ADC_InjectedChannel_1) || \
00439 ((CHANNEL) == ADC_InjectedChannel_2) || \
00440 ((CHANNEL) == ADC_InjectedChannel_3) || \
00441 ((CHANNEL) == ADC_InjectedChannel_4))
00450 #define ADC_AnalogWatchdog_SingleRegEnable ((uint32_t)0x00800200)
00451 #define ADC_AnalogWatchdog_SingleInjecEnable ((uint32_t)0x00400200)
00452 #define ADC_AnalogWatchdog_SingleRegOrInjecEnable ((uint32_t)0x00C00200)
00453 #define ADC_AnalogWatchdog_AllRegEnable ((uint32_t)0x00800000)
00454 #define ADC_AnalogWatchdog_AllInjecEnable ((uint32_t)0x00400000)
00455 #define ADC_AnalogWatchdog_AllRegAllInjecEnable ((uint32_t)0x00C00000)
00456 #define ADC_AnalogWatchdog_None ((uint32_t)0x00000000)
00457 #define IS_ADC_ANALOG_WATCHDOG(WATCHDOG) (((WATCHDOG) == ADC_AnalogWatchdog_SingleRegEnable) || \
00458 ((WATCHDOG) == ADC_AnalogWatchdog_SingleInjecEnable) || \
00459 ((WATCHDOG) == ADC_AnalogWatchdog_SingleRegOrInjecEnable) || \
00460 ((WATCHDOG) == ADC_AnalogWatchdog_AllRegEnable) || \
00461 ((WATCHDOG) == ADC_AnalogWatchdog_AllInjecEnable) || \
00462 ((WATCHDOG) == ADC_AnalogWatchdog_AllRegAllInjecEnable) || \
00463 ((WATCHDOG) == ADC_AnalogWatchdog_None))
00472 #define ADC_IT_EOC ((uint16_t)0x0205)
00473 #define ADC_IT_AWD ((uint16_t)0x0106)
00474 #define ADC_IT_JEOC ((uint16_t)0x0407)
00475 #define ADC_IT_OVR ((uint16_t)0x201A)
00476 #define IS_ADC_IT(IT) (((IT) == ADC_IT_EOC) || ((IT) == ADC_IT_AWD) || \
00477 ((IT) == ADC_IT_JEOC) || ((IT) == ADC_IT_OVR))
00486 #define ADC_FLAG_AWD ((uint8_t)0x01)
00487 #define ADC_FLAG_EOC ((uint8_t)0x02)
00488 #define ADC_FLAG_JEOC ((uint8_t)0x04)
00489 #define ADC_FLAG_JSTRT ((uint8_t)0x08)
00490 #define ADC_FLAG_STRT ((uint8_t)0x10)
00491 #define ADC_FLAG_OVR ((uint8_t)0x20)
00492
00493 #define IS_ADC_CLEAR_FLAG(FLAG) (((FLAG) & (uint8_t)0xC0) == 0x00) && ((FLAG) != 0x00)
00494 #define IS_ADC_GET_FLAG(FLAG) (((FLAG) == ADC_FLAG_AWD) || \
00495 ((FLAG) == ADC_FLAG_EOC) || \
00496 ((FLAG) == ADC_FLAG_JEOC) || \
00497 ((FLAG) == ADC_FLAG_JSTRT) || \
00498 ((FLAG) == ADC_FLAG_STRT) || \
00499 ((FLAG) == ADC_FLAG_OVR))
00508 #define IS_ADC_THRESHOLD(THRESHOLD) ((THRESHOLD) <= 0xFFFF)
00517 #define IS_ADC_OFFSET(OFFSET) ((OFFSET) <= 0xFFFF)

```



```

00526 #define IS_ADC_INJECTED_LENGTH(LENGTH) (((LENGTH) >= 0x1) && ((LENGTH) <= 0x4))
00535 #define IS_ADC_INJECTED_RANK(RANK) (((RANK) >= 0x1) && ((RANK) <= 0x4))
00544 #define IS_ADC_REGULAR_LENGTH(LENGTH) (((LENGTH) >= 0x1) && ((LENGTH) <= 0x10))
00553 #define IS_ADC_REGULAR_RANK(RANK) (((RANK) >= 0x1) && ((RANK) <= 0x10))
00562 #define IS_ADC_REGULAR_DISC_NUMBER(NUMBER) (((NUMBER) >= 0x1) && ((NUMBER) <= 0x8))
00572 /* Exported macro -----*/
00573 /* Exported functions -----*/
00574
00575 /* Function used to set the ADC configuration to the default reset state *****/
00576 void ADC_DeInit(void);
00577
00578 /* Initialization and Configuration functions *****/
00579 void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct);
00580 void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct);
00581 void ADC_CommonInit(ADC_CommonInitTypeDef* ADC_CommonInitStruct);
00582 void ADC_CommonStructInit(ADC_CommonInitTypeDef* ADC_CommonInitStruct);
00583 void ADC_Cmd(ADC_TypeDef* ADCx, FunctionalState NewState);
00584
00585 /* Analog Watchdog configuration functions *****/
00586 void ADC_AnalogWatchdogCmd(ADC_TypeDef* ADCx, uint32_t ADC_AnalogWatchdog);
00587 void ADC_AnalogWatchdogThresholdsConfig(ADC_TypeDef* ADCx, uint16_t HighThreshold, uint16_t
LowThreshold);
00588 void ADC_AnalogWatchdogSingleChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel);
00589
00590 /* Temperature Sensor, Vrefint and VBAT management functions *****/
00591 void ADC_TempSensorVrefintCmd(FunctionalState NewState);
00592 void ADC_VBATCmd(FunctionalState NewState);
00593
00594 /* Regular Channels Configuration functions *****/
00595 void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t
ADC_SampleTime);
00596 void ADC_SoftwareStartConv(ADC_TypeDef* ADCx);
00597 FlagStatus ADC_GetSoftwareStartConvStatus(ADC_TypeDef* ADCx);
00598 void ADC_EOCOnEachRegularChannelCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
00599 void ADC_ContinuousModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
00600 void ADC_DiscModeChannelCountConfig(ADC_TypeDef* ADCx, uint8_t Number);
00601 void ADC_DiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
00602 uint16_t ADC_GetConversionValue(ADC_TypeDef* ADCx);
00603 uint32_t ADC_GetMultiModeConversionValue(void);
00604
00605 /* Regular Channels DMA Configuration functions *****/
00606 void ADC_DMACmd(ADC_TypeDef* ADCx, FunctionalState NewState);
00607 void ADC_DMARequestAfterLastTransferCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
00608 void ADC_MultiModeDMARequestAfterLastTransferCmd(FunctionalState NewState);
00609
00610 /* Injected channels Configuration functions *****/
00611 void ADC_InjectedChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t
ADC_SampleTime);
00612 void ADC_InjectedSequencerLengthConfig(ADC_TypeDef* ADCx, uint8_t Length);
00613 void ADC_SetInjectedOffset(ADC_TypeDef* ADCx, uint8_t ADC_InjectedChannel, uint16_t Offset);
00614 void ADC_ExternalTrigInjectedConvConfig(ADC_TypeDef* ADCx, uint32_t ADC_ExternalTrigInjecConv);
00615 void ADC_ExternalTrigInjectedConvEdgeConfig(ADC_TypeDef* ADCx, uint32_t
ADC_ExternalTrigInjecConvEdge);
00616 void ADC_SoftwareStartInjectedConv(ADC_TypeDef* ADCx);
00617 FlagStatus ADC_GetSoftwareStartInjectedConvCmdStatus(ADC_TypeDef* ADCx);
00618 void ADC_AutoInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
00619 void ADC_InjectedDiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
00620 uint16_t ADC_GetInjectedConversionValue(ADC_TypeDef* ADCx, uint8_t ADC_InjectedChannel);
00621
00622 /* Interrupts and flags management functions *****/
00623 void ADC_ITConfig(ADC_TypeDef* ADCx, uint16_t ADC_IT, FunctionalState NewState);
00624 FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, uint8_t ADC_FLAG);
00625 void ADC_ClearFlag(ADC_TypeDef* ADCx, uint8_t ADC_FLAG);
00626 ITStatus ADC_GetITStatus(ADC_TypeDef* ADCx, uint16_t ADC_IT);
00627 void ADC_ClearITPendingBit(ADC_TypeDef* ADCx, uint16_t ADC_IT);
00628
00629 #ifdef __cplusplus
00630 }
00631 #endif
00632
00633 #endif /* __STM32F4xx_ADC_H */
00634
00643 /***** (C) COPYRIGHT 2011 STMicroelectronics *****/

```

6.18 drivers/stm32f4xx_gpio.c File Reference

This file provides firmware functions to manage the following functionalities of the GPIO peripheral:

```

#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"

```

Functions

- void [GPIO_DeInit](#) (GPIO_TypeDef *GPIOx)
Deinitializes the GPIOx peripheral registers to their default reset values.
- void [GPIO_Init](#) (GPIO_TypeDef *GPIOx, [GPIO_InitTypeDef](#) *GPIO_InitStruct)
Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
- void [GPIO_StructInit](#) ([GPIO_InitTypeDef](#) *GPIO_InitStruct)
Fills each GPIO_InitStruct member with its default value.
- void [GPIO_PinLockConfig](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Locks GPIO Pins configuration registers.
- uint8_t [GPIO_ReadInputDataBit](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Reads the specified input port pin.
- uint16_t [GPIO_ReadInputData](#) (GPIO_TypeDef *GPIOx)
Reads the specified GPIO input data port.
- uint8_t [GPIO_ReadOutputDataBit](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Reads the specified output data port bit.
- uint16_t [GPIO_ReadOutputData](#) (GPIO_TypeDef *GPIOx)
Reads the specified GPIO output data port.
- void [GPIO_SetBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Sets the selected data port bits.
- void [GPIO_ResetBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Clears the selected data port bits.
- void [GPIO_WriteBit](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, [BitAction](#) BitVal)
Sets or clears the selected data port bit.
- void [GPIO_Write](#) (GPIO_TypeDef *GPIOx, uint16_t PortVal)
Writes data to the specified GPIO data port.
- void [GPIO_ToggleBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Toggles the specified GPIO pins..
- void [GPIO_PinAFConfig](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_PinSource, uint8_t GPIO_AF)
Changes the mapping of the specified pin.

6.18.1 Detailed Description

This file provides firmware functions to manage the following functionalities of the GPIO peripheral:

Author

MCD Application Team

Version

V1.0.0

Date

30-September-2011

- Initialization and Configuration
- GPIO Read and Write
- GPIO Alternate functions configuration

```

*
* =====
*                               How to use this driver
*                               =====
*
* 1. Enable the GPIO AHB clock using the following function
*     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOx, ENABLE);
*
* 2. Configure the GPIO pin(s) using GPIO_Init()
*     Four possible configuration are available for each pin:
*         - Input: Floating, Pull-up, Pull-down.
*         - Output: Push-Pull (Pull-up, Pull-down or no Pull)
*                   Open Drain (Pull-up, Pull-down or no Pull).
*         In output mode, the speed is configurable: 2 MHz, 25 MHz,
*         50 MHz or 100 MHz.
*         - Alternate Function: Push-Pull (Pull-up, Pull-down or no Pull)
*                   Open Drain (Pull-up, Pull-down or no Pull).
*         - Analog: required mode when a pin is to be used as ADC channel
*                   or DAC output.
*
* 3- Peripherals alternate function:
*     - For ADC and DAC, configure the desired pin in analog mode using
*       GPIO_InitStruct->GPIO_Mode = GPIO_Mode_AN;
*     - For other peripherals (TIM, USART...):
*         - Connect the pin to the desired peripherals' Alternate
*           Function (AF) using GPIO_PinAFConfig() function
*         - Configure the desired pin in alternate function mode using
*           GPIO_InitStruct->GPIO_Mode = GPIO_Mode_AF
*         - Select the type, pull-up/pull-down and output speed via
*           GPIO_PuPd, GPIO_OType and GPIO_Speed members
*         - Call GPIO_Init() function
*
* 4. To get the level of a pin configured in input mode use GPIO_ReadInputDataBit()
*
* 5. To set/reset the level of a pin configured in output mode use
*     GPIO_SetBits()/GPIO_ResetBits()
*
* 6. During and just after reset, the alternate functions are not
*     active and the GPIO pins are configured in input floating mode
*     (except JTAG pins).
*
* 7. 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.
*
* 8. 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.
*
*

```

Attention

THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

© COPYRIGHT 2011 STMicroelectronics

6.19 drivers/stm32f4xx_gpio.h File Reference

This file contains all the functions prototypes for the GPIO firmware library.

```
#include "stm32f4xx.h"
```

Data Structures

- struct [GPIO_InitTypeDef](#)
GPIO Init structure definition

Macros

- `#define IS_GPIO_ALL_PERIPH(PERIPH)`
- `#define IS_GPIO_MODE(MODE)`
- `#define IS_GPIO_OTYPE(OTYPE) (((OTYPE) == GPIO_OType_PP) || ((OTYPE) == GPIO_OType_OD))`
- `#define IS_GPIO_SPEED(SPEED)`
- `#define IS_GPIO_PUPD(PUPD)`
- `#define IS_GPIO_BIT_ACTION(ACTION) (((ACTION) == Bit_RESET) || ((ACTION) == Bit_SET))`
- `#define GPIO_Pin_0 ((uint16_t)0x0001) /* Pin 0 selected */`
- `#define GPIO_Pin_1 ((uint16_t)0x0002) /* Pin 1 selected */`
- `#define GPIO_Pin_2 ((uint16_t)0x0004) /* Pin 2 selected */`
- `#define GPIO_Pin_3 ((uint16_t)0x0008) /* Pin 3 selected */`
- `#define GPIO_Pin_4 ((uint16_t)0x0010) /* Pin 4 selected */`
- `#define GPIO_Pin_5 ((uint16_t)0x0020) /* Pin 5 selected */`
- `#define GPIO_Pin_6 ((uint16_t)0x0040) /* Pin 6 selected */`
- `#define GPIO_Pin_7 ((uint16_t)0x0080) /* Pin 7 selected */`
- `#define GPIO_Pin_8 ((uint16_t)0x0100) /* Pin 8 selected */`
- `#define GPIO_Pin_9 ((uint16_t)0x0200) /* Pin 9 selected */`
- `#define GPIO_Pin_10 ((uint16_t)0x0400) /* Pin 10 selected */`
- `#define GPIO_Pin_11 ((uint16_t)0x0800) /* Pin 11 selected */`
- `#define GPIO_Pin_12 ((uint16_t)0x1000) /* Pin 12 selected */`
- `#define GPIO_Pin_13 ((uint16_t)0x2000) /* Pin 13 selected */`
- `#define GPIO_Pin_14 ((uint16_t)0x4000) /* Pin 14 selected */`
- `#define GPIO_Pin_15 ((uint16_t)0x8000) /* Pin 15 selected */`
- `#define GPIO_Pin_All ((uint16_t)0xFFFF) /* All pins selected */`
- `#define IS_GPIO_PIN(PIN) (((PIN) & (uint16_t)0x00) == 0x00) && ((PIN) != (uint16_t)0x00)`
- `#define IS_GET_GPIO_PIN(PIN)`
- `#define GPIO_PinSource0 ((uint8_t)0x00)`
- `#define GPIO_PinSource1 ((uint8_t)0x01)`
- `#define GPIO_PinSource2 ((uint8_t)0x02)`
- `#define GPIO_PinSource3 ((uint8_t)0x03)`
- `#define GPIO_PinSource4 ((uint8_t)0x04)`
- `#define GPIO_PinSource5 ((uint8_t)0x05)`
- `#define GPIO_PinSource6 ((uint8_t)0x06)`
- `#define GPIO_PinSource7 ((uint8_t)0x07)`
- `#define GPIO_PinSource8 ((uint8_t)0x08)`
- `#define GPIO_PinSource9 ((uint8_t)0x09)`
- `#define GPIO_PinSource10 ((uint8_t)0x0A)`
- `#define GPIO_PinSource11 ((uint8_t)0x0B)`
- `#define GPIO_PinSource12 ((uint8_t)0x0C)`
- `#define GPIO_PinSource13 ((uint8_t)0x0D)`
- `#define GPIO_PinSource14 ((uint8_t)0x0E)`
- `#define GPIO_PinSource15 ((uint8_t)0x0F)`
- `#define IS_GPIO_PIN_SOURCE(PINSOURCE)`
- `#define GPIO_AF_RTC_50Hz ((uint8_t)0x00) /* RTC_50Hz Alternate Function mapping */`
AF 0 selection
- `#define GPIO_AF_MCO ((uint8_t)0x00) /* MCO (MCO1 and MCO2) Alternate Function mapping */`

- #define [GPIO_AF_TAMPER](#) ((uint8_t)0x00) /* TAMPER (TAMPER_1 and TAMPER_2) Alternate Function mapping */
- #define [GPIO_AF_SWJ](#) ((uint8_t)0x00) /* SWJ (SWD and JTAG) Alternate Function mapping */
- #define [GPIO_AF_TRACE](#) ((uint8_t)0x00) /* TRACE Alternate Function mapping */
- #define [GPIO_AF_TIM1](#) ((uint8_t)0x01) /* TIM1 Alternate Function mapping */
- AF 1 selection*
- #define [GPIO_AF_TIM2](#) ((uint8_t)0x01) /* TIM2 Alternate Function mapping */
- #define [GPIO_AF_TIM3](#) ((uint8_t)0x02) /* TIM3 Alternate Function mapping */
- AF 2 selection*
- #define [GPIO_AF_TIM4](#) ((uint8_t)0x02) /* TIM4 Alternate Function mapping */
- #define [GPIO_AF_TIM5](#) ((uint8_t)0x02) /* TIM5 Alternate Function mapping */
- #define [GPIO_AF_TIM8](#) ((uint8_t)0x03) /* TIM8 Alternate Function mapping */
- AF 3 selection*
- #define [GPIO_AF_TIM9](#) ((uint8_t)0x03) /* TIM9 Alternate Function mapping */
- #define [GPIO_AF_TIM10](#) ((uint8_t)0x03) /* TIM10 Alternate Function mapping */
- #define [GPIO_AF_TIM11](#) ((uint8_t)0x03) /* TIM11 Alternate Function mapping */
- #define [GPIO_AF_I2C1](#) ((uint8_t)0x04) /* I2C1 Alternate Function mapping */
- AF 4 selection*
- #define [GPIO_AF_I2C2](#) ((uint8_t)0x04) /* I2C2 Alternate Function mapping */
- #define [GPIO_AF_I2C3](#) ((uint8_t)0x04) /* I2C3 Alternate Function mapping */
- #define [GPIO_AF_SPI1](#) ((uint8_t)0x05) /* SPI1 Alternate Function mapping */
- AF 5 selection*
- #define [GPIO_AF_SPI2](#) ((uint8_t)0x05) /* SPI2/I2S2 Alternate Function mapping */
- #define [GPIO_AF_SPI3](#) ((uint8_t)0x06) /* SPI3/I2S3 Alternate Function mapping */
- AF 6 selection*
- #define [GPIO_AF_USART1](#) ((uint8_t)0x07) /* USART1 Alternate Function mapping */
- AF 7 selection*
- #define [GPIO_AF_USART2](#) ((uint8_t)0x07) /* USART2 Alternate Function mapping */
- #define [GPIO_AF_USART3](#) ((uint8_t)0x07) /* USART3 Alternate Function mapping */
- #define [GPIO_AF_I2S3ext](#) ((uint8_t)0x07) /* I2S3ext Alternate Function mapping */
- #define [GPIO_AF_UART4](#) ((uint8_t)0x08) /* UART4 Alternate Function mapping */
- AF 8 selection*
- #define [GPIO_AF_UART5](#) ((uint8_t)0x08) /* UART5 Alternate Function mapping */
- #define [GPIO_AF_USART6](#) ((uint8_t)0x08) /* USART6 Alternate Function mapping */
- #define [GPIO_AF_CAN1](#) ((uint8_t)0x09) /* CAN1 Alternate Function mapping */
- AF 9 selection.*
- #define [GPIO_AF_CAN2](#) ((uint8_t)0x09) /* CAN2 Alternate Function mapping */
- #define [GPIO_AF_TIM12](#) ((uint8_t)0x09) /* TIM12 Alternate Function mapping */
- #define [GPIO_AF_TIM13](#) ((uint8_t)0x09) /* TIM13 Alternate Function mapping */
- #define [GPIO_AF_TIM14](#) ((uint8_t)0x09) /* TIM14 Alternate Function mapping */
- #define [GPIO_AF_OTG_FS](#) ((uint8_t)0xA) /* OTG_FS Alternate Function mapping */
- AF 10 selection*
- #define [GPIO_AF_OTG_HS](#) ((uint8_t)0xA) /* OTG_HS Alternate Function mapping */
- #define [GPIO_AF_ETH](#) ((uint8_t)0x0B) /* ETHERNET Alternate Function mapping */
- AF 11 selection*

- `#define GPIO_AF_FSMC ((uint8_t)0xC) /* FSMC Alternate Function mapping */`
AF 12 selection
- `#define GPIO_AF_OTG_HS_FS ((uint8_t)0xC) /* OTG HS configured in FS, Alternate Function mapping */`
- `#define GPIO_AF_SDIO ((uint8_t)0xC) /* SDIO Alternate Function mapping */`
- `#define GPIO_AF_DCMI ((uint8_t)0x0D) /* DCMI Alternate Function mapping */`
AF 13 selection
- `#define GPIO_AF_EVENTOUT ((uint8_t)0x0F) /* EVENTOUT Alternate Function mapping */`
AF 15 selection
- `#define IS_GPIO_AF(AF)`
- `#define GPIO_Mode_AIN GPIO_Mode_AN`
- `#define GPIO_AF_OTG1_FS GPIO_AF_OTG_FS`
- `#define GPIO_AF_OTG2_HS GPIO_AF_OTG_HS`
- `#define GPIO_AF_OTG2_FS GPIO_AF_OTG_HS_FS`

Enumerations

- enum `GPIO_Mode_TypeDef` { `GPIO_Mode_IN` = 0x00 , `GPIO_Mode_OUT` = 0x01 , `GPIO_Mode_AF` = 0x02 , `GPIO_Mode_AN` = 0x03 }
- GPIO Configuration Mode enumeration.*
- enum `GPIO_OType_TypeDef` { `GPIO_OType_PP` = 0x00 , `GPIO_OType_OD` = 0x01 }
- GPIO Output type enumeration.*
- enum `GPIO_Speed_TypeDef` { `GPIO_Speed_2MHz` = 0x00 , `GPIO_Speed_25MHz` = 0x01 , `GPIO_Speed_50MHz` = 0x02 , `GPIO_Speed_100MHz` = 0x03 }
- GPIO Output Maximum frequency enumeration.*
- enum `GPIO_PuPd_TypeDef` { `GPIO_PuPd_NOPULL` = 0x00 , `GPIO_PuPd_UP` = 0x01 , `GPIO_PuPd_DOWN` = 0x02 }
- GPIO Configuration PullUp PullDown enumeration.*
- enum `BitAction` { `Bit_RESET` = 0 , `Bit_SET` }
- GPIO Bit SET and Bit RESET enumeration.*

Functions

- void `GPIO_DeInit` (GPIO_TypeDef *GPIOx)
Deinitializes the GPIOx peripheral registers to their default reset values.
- void `GPIO_Init` (GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct)
Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
- void `GPIO_StructInit` (GPIO_InitTypeDef *GPIO_InitStruct)
Fills each GPIO_InitStruct member with its default value.
- void `GPIO_PinLockConfig` (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Locks GPIO Pins configuration registers.
- uint8_t `GPIO_ReadInputDataBit` (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Reads the specified input port pin.
- uint16_t `GPIO_ReadInputData` (GPIO_TypeDef *GPIOx)
Reads the specified GPIO input data port.
- uint8_t `GPIO_ReadOutputDataBit` (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Reads the specified output data port bit.
- uint16_t `GPIO_ReadOutputData` (GPIO_TypeDef *GPIOx)
Reads the specified GPIO output data port.
- void `GPIO_SetBits` (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Sets the selected data port bits.

- void [GPIO_ResetBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Clears the selected data port bits.
- void [GPIO_WriteBit](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, [BitAction](#) BitVal)
Sets or clears the selected data port bit.
- void [GPIO_Write](#) (GPIO_TypeDef *GPIOx, uint16_t PortVal)
Writes data to the specified GPIO data port.
- void [GPIO_ToggleBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
Toggles the specified GPIO pins..
- void [GPIO_PinAFConfig](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_PinSource, uint8_t GPIO_AF)
Changes the mapping of the specified pin.

6.19.1 Detailed Description

This file contains all the functions prototypes for the GPIO firmware library.

Author

MCD Application Team

Version

V1.0.0

Date

30-September-2011

Attention

THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

© COPYRIGHT 2011 STMicroelectronics

6.20 stm32f4xx_gpio.h

[Go to the documentation of this file.](#)

```
00001
00023 /* Define to prevent recursive inclusion -----*/
00024 #ifndef __STM32F4xx_GPIO_H
00025 #define __STM32F4xx_GPIO_H
00026
00027 #ifdef __cplusplus
00028     extern "C" {
00029 #endif
00030
00031 /* Includes -----*/
00032 #include "stm32f4xx.h"
00033
00042 /* Exported types -----*/
00043
00044 #define IS_GPIO_ALL_PERIPH(PERIPH) (( (PERIPH) == GPIOA) || \
00045                                     ( (PERIPH) == GPIOB) || \
00046                                     ( (PERIPH) == GPIOC) || \
00047                                     ( (PERIPH) == GPIOD) || \
00048                                     ( (PERIPH) == GPIOE) || \
00049                                     ( (PERIPH) == GPIOF) || \
```

```

00050                                     ((PERIPH) == GPIOG) || \
00051                                     ((PERIPH) == GPIOH) || \
00052                                     ((PERIPH) == GPIOI))
00053
00057 typedef enum
00058 {
00059     GPIO_Mode_IN    = 0x00,
00060     GPIO_Mode_OUT   = 0x01,
00061     GPIO_Mode_AF    = 0x02,
00062     GPIO_Mode_AN    = 0x03
00063 }GPIOMode_TypeDef;
00064 #define IS_GPIO_MODE(MODE) (((MODE) == GPIO_Mode_IN) || ((MODE) == GPIO_Mode_OUT) || \
00065                             ((MODE) == GPIO_Mode_AF) || ((MODE) == GPIO_Mode_AN))
00066
00070 typedef enum
00071 {
00072     GPIO_OType_PP = 0x00,
00073     GPIO_OType_OD = 0x01
00074 }GPIOType_TypeDef;
00075 #define IS_GPIO_OTYPE(OTYPE) (((OTYPE) == GPIO_OType_PP) || ((OTYPE) == GPIO_OType_OD))
00076
00077
00081 typedef enum
00082 {
00083     GPIO_Speed_2MHz   = 0x00,
00084     GPIO_Speed_25MHz  = 0x01,
00085     GPIO_Speed_50MHz  = 0x02,
00086     GPIO_Speed_100MHz = 0x03
00087 }GPIOSpeed_TypeDef;
00088 #define IS_GPIO_SPEED(SPEED) (((SPEED) == GPIO_Speed_2MHz) || ((SPEED) == GPIO_Speed_25MHz) || \
00089                               ((SPEED) == GPIO_Speed_50MHz) || ((SPEED) == GPIO_Speed_100MHz))
00090
00094 typedef enum
00095 {
00096     GPIO_PuPd_NOPULL = 0x00,
00097     GPIO_PuPd_UP     = 0x01,
00098     GPIO_PuPd_DOWN   = 0x02
00099 }GPIOPuPd_TypeDef;
00100 #define IS_GPIO_PUPD(PUPD) (((PUPD) == GPIO_PuPd_NOPULL) || ((PUPD) == GPIO_PuPd_UP) || \
00101                             ((PUPD) == GPIO_PuPd_DOWN))
00102
00106 typedef enum
00107 {
00108     Bit_RESET = 0,
00109     Bit_SET
00110 }BitAction;
00111 #define IS_GPIO_BIT_ACTION(ACTION) (((ACTION) == Bit_RESET) || ((ACTION) == Bit_SET))
00112
00113
00117 typedef struct
00118 {
00119     uint32_t GPIO_Pin;
00120     GPIOMode_TypeDef GPIO_Mode;
00121     GPIOSpeed_TypeDef GPIO_Speed;
00122     GPIOType_TypeDef GPIO_OType;
00123     GPIOPuPd_TypeDef GPIO_PuPd;
00124 }GPIO_InitTypeDef;
00125
00135 /* Exported constants -----*/
00136
00144 #define GPIO_Pin_0          ((uint16_t)0x0001) /* Pin 0 selected */
00145 #define GPIO_Pin_1          ((uint16_t)0x0002) /* Pin 1 selected */
00146 #define GPIO_Pin_2          ((uint16_t)0x0004) /* Pin 2 selected */
00147 #define GPIO_Pin_3          ((uint16_t)0x0008) /* Pin 3 selected */
00148 #define GPIO_Pin_4          ((uint16_t)0x0010) /* Pin 4 selected */
00149 #define GPIO_Pin_5          ((uint16_t)0x0020) /* Pin 5 selected */
00150 #define GPIO_Pin_6          ((uint16_t)0x0040) /* Pin 6 selected */
00151 #define GPIO_Pin_7          ((uint16_t)0x0080) /* Pin 7 selected */
00152 #define GPIO_Pin_8          ((uint16_t)0x0100) /* Pin 8 selected */
00153 #define GPIO_Pin_9          ((uint16_t)0x0200) /* Pin 9 selected */
00154 #define GPIO_Pin_10         ((uint16_t)0x0400) /* Pin 10 selected */
00155 #define GPIO_Pin_11         ((uint16_t)0x0800) /* Pin 11 selected */
00156 #define GPIO_Pin_12         ((uint16_t)0x1000) /* Pin 12 selected */
00157 #define GPIO_Pin_13         ((uint16_t)0x2000) /* Pin 13 selected */
00158 #define GPIO_Pin_14         ((uint16_t)0x4000) /* Pin 14 selected */
00159 #define GPIO_Pin_15         ((uint16_t)0x8000) /* Pin 15 selected */
00160 #define GPIO_Pin_All        ((uint16_t)0xFFFF) /* All pins selected */
00161
00162 #define IS_GPIO_PIN(PIN) (((PIN) & (uint16_t)0x00) == 0x00) && ((PIN) != (uint16_t)0x00)
00163 #define IS_GET_GPIO_PIN(PIN) (((PIN) == GPIO_Pin_0) || \
00164                               ((PIN) == GPIO_Pin_1) || \
00165                               ((PIN) == GPIO_Pin_2) || \
00166                               ((PIN) == GPIO_Pin_3) || \
00167                               ((PIN) == GPIO_Pin_4) || \
00168                               ((PIN) == GPIO_Pin_5) || \
00169                               ((PIN) == GPIO_Pin_6) || \
00170                               ((PIN) == GPIO_Pin_7) || \

```



```

00171 ((PIN) == GPIO_Pin_8) || \
00172 ((PIN) == GPIO_Pin_9) || \
00173 ((PIN) == GPIO_Pin_10) || \
00174 ((PIN) == GPIO_Pin_11) || \
00175 ((PIN) == GPIO_Pin_12) || \
00176 ((PIN) == GPIO_Pin_13) || \
00177 ((PIN) == GPIO_Pin_14) || \
00178 ((PIN) == GPIO_Pin_15))
00187 #define GPIO_PinSource0 ((uint8_t)0x00)
00188 #define GPIO_PinSource1 ((uint8_t)0x01)
00189 #define GPIO_PinSource2 ((uint8_t)0x02)
00190 #define GPIO_PinSource3 ((uint8_t)0x03)
00191 #define GPIO_PinSource4 ((uint8_t)0x04)
00192 #define GPIO_PinSource5 ((uint8_t)0x05)
00193 #define GPIO_PinSource6 ((uint8_t)0x06)
00194 #define GPIO_PinSource7 ((uint8_t)0x07)
00195 #define GPIO_PinSource8 ((uint8_t)0x08)
00196 #define GPIO_PinSource9 ((uint8_t)0x09)
00197 #define GPIO_PinSource10 ((uint8_t)0x0A)
00198 #define GPIO_PinSource11 ((uint8_t)0x0B)
00199 #define GPIO_PinSource12 ((uint8_t)0x0C)
00200 #define GPIO_PinSource13 ((uint8_t)0x0D)
00201 #define GPIO_PinSource14 ((uint8_t)0x0E)
00202 #define GPIO_PinSource15 ((uint8_t)0x0F)
00203
00204 #define IS_GPIO_PIN_SOURCE(PINSOURCE) (((PINSOURCE) == GPIO_PinSource0) || \
00205 ((PINSOURCE) == GPIO_PinSource1) || \
00206 ((PINSOURCE) == GPIO_PinSource2) || \
00207 ((PINSOURCE) == GPIO_PinSource3) || \
00208 ((PINSOURCE) == GPIO_PinSource4) || \
00209 ((PINSOURCE) == GPIO_PinSource5) || \
00210 ((PINSOURCE) == GPIO_PinSource6) || \
00211 ((PINSOURCE) == GPIO_PinSource7) || \
00212 ((PINSOURCE) == GPIO_PinSource8) || \
00213 ((PINSOURCE) == GPIO_PinSource9) || \
00214 ((PINSOURCE) == GPIO_PinSource10) || \
00215 ((PINSOURCE) == GPIO_PinSource11) || \
00216 ((PINSOURCE) == GPIO_PinSource12) || \
00217 ((PINSOURCE) == GPIO_PinSource13) || \
00218 ((PINSOURCE) == GPIO_PinSource14) || \
00219 ((PINSOURCE) == GPIO_PinSource15))
00230 #define GPIO_AF_RTC_50Hz ((uint8_t)0x00) /* RTC_50Hz Alternate Function mapping */
00231 #define GPIO_AF_MCO ((uint8_t)0x00) /* MCO (MCO1 and MCO2) Alternate Function mapping */
00232 #define GPIO_AF_TAMPER ((uint8_t)0x00) /* TAMPER (TAMPER_1 and TAMPER_2) Alternate Function mapping */
00233 #define GPIO_AF_SWJ ((uint8_t)0x00) /* SWJ (SWD and JTAG) Alternate Function mapping */
00234 #define GPIO_AF_TRACE ((uint8_t)0x00) /* TRACE Alternate Function mapping */
00235
00239 #define GPIO_AF_TIM1 ((uint8_t)0x01) /* TIM1 Alternate Function mapping */
00240 #define GPIO_AF_TIM2 ((uint8_t)0x01) /* TIM2 Alternate Function mapping */
00241
00245 #define GPIO_AF_TIM3 ((uint8_t)0x02) /* TIM3 Alternate Function mapping */
00246 #define GPIO_AF_TIM4 ((uint8_t)0x02) /* TIM4 Alternate Function mapping */
00247 #define GPIO_AF_TIM5 ((uint8_t)0x02) /* TIM5 Alternate Function mapping */
00248
00252 #define GPIO_AF_TIM8 ((uint8_t)0x03) /* TIM8 Alternate Function mapping */
00253 #define GPIO_AF_TIM9 ((uint8_t)0x03) /* TIM9 Alternate Function mapping */
00254 #define GPIO_AF_TIM10 ((uint8_t)0x03) /* TIM10 Alternate Function mapping */
00255 #define GPIO_AF_TIM11 ((uint8_t)0x03) /* TIM11 Alternate Function mapping */
00256
00260 #define GPIO_AF_I2C1 ((uint8_t)0x04) /* I2C1 Alternate Function mapping */
00261 #define GPIO_AF_I2C2 ((uint8_t)0x04) /* I2C2 Alternate Function mapping */
00262 #define GPIO_AF_I2C3 ((uint8_t)0x04) /* I2C3 Alternate Function mapping */
00263
00267 #define GPIO_AF_SPI1 ((uint8_t)0x05) /* SPI1 Alternate Function mapping */
00268 #define GPIO_AF_SPI2 ((uint8_t)0x05) /* SPI2/I2S2 Alternate Function mapping */
00269
00273 #define GPIO_AF_SPI3 ((uint8_t)0x06) /* SPI3/I2S3 Alternate Function mapping */
00274
00278 #define GPIO_AF_USART1 ((uint8_t)0x07) /* USART1 Alternate Function mapping */
00279 #define GPIO_AF_USART2 ((uint8_t)0x07) /* USART2 Alternate Function mapping */
00280 #define GPIO_AF_USART3 ((uint8_t)0x07) /* USART3 Alternate Function mapping */
00281 #define GPIO_AF_I2S3ext ((uint8_t)0x07) /* I2S3ext Alternate Function mapping */
00282
00286 #define GPIO_AF_UART4 ((uint8_t)0x08) /* UART4 Alternate Function mapping */
00287 #define GPIO_AF_UART5 ((uint8_t)0x08) /* UART5 Alternate Function mapping */
00288 #define GPIO_AF_USART6 ((uint8_t)0x08) /* USART6 Alternate Function mapping */
00289
00293 #define GPIO_AF_CAN1 ((uint8_t)0x09) /* CAN1 Alternate Function mapping */
00294 #define GPIO_AF_CAN2 ((uint8_t)0x09) /* CAN2 Alternate Function mapping */
00295 #define GPIO_AF_TIM12 ((uint8_t)0x09) /* TIM12 Alternate Function mapping */
00296 #define GPIO_AF_TIM13 ((uint8_t)0x09) /* TIM13 Alternate Function mapping */
00297 #define GPIO_AF_TIM14 ((uint8_t)0x09) /* TIM14 Alternate Function mapping */
00298
00302 #define GPIO_AF_OTG_FS ((uint8_t)0xA) /* OTG_FS Alternate Function mapping */
00303 #define GPIO_AF_OTG_HS ((uint8_t)0xA) /* OTG_HS Alternate Function mapping */
00304

```

```

00308 #define GPIO_AF_ETH                ((uint8_t)0x0B) /* ETHERNET Alternate Function mapping */
00309
00313 #define GPIO_AF_FSMC                ((uint8_t)0xC) /* FSMC Alternate Function mapping */
00314 #define GPIO_AF_OTG_HS_FS          ((uint8_t)0xC) /* OTG HS configured in FS, Alternate Function mapping
*/
00315 #define GPIO_AF_SDIO                ((uint8_t)0xC) /* SDIO Alternate Function mapping */
00316
00320 #define GPIO_AF_DCMCI                ((uint8_t)0x0D) /* DCMCI Alternate Function mapping */
00321
00325 #define GPIO_AF_EVENTOUT            ((uint8_t)0x0F) /* EVENTOUT Alternate Function mapping */
00326
00327 #define IS_GPIO_AF(AF)              (((AF) == GPIO_AF_RTC_50Hz) || ((AF) == GPIO_AF_TIM14) || \
00328                                     ((AF) == GPIO_AF_MCO) || ((AF) == GPIO_AF_TAMPER) || \
00329                                     ((AF) == GPIO_AF_SWJ) || ((AF) == GPIO_AF_TRACE) || \
00330                                     ((AF) == GPIO_AF_TIM1) || ((AF) == GPIO_AF_TIM2) || \
00331                                     ((AF) == GPIO_AF_TIM3) || ((AF) == GPIO_AF_TIM4) || \
00332                                     ((AF) == GPIO_AF_TIM5) || ((AF) == GPIO_AF_TIM8) || \
00333                                     ((AF) == GPIO_AF_I2C1) || ((AF) == GPIO_AF_I2C2) || \
00334                                     ((AF) == GPIO_AF_I2C3) || ((AF) == GPIO_AF_SPI1) || \
00335                                     ((AF) == GPIO_AF_SPI2) || ((AF) == GPIO_AF_TIM13) || \
00336                                     ((AF) == GPIO_AF_SPI3) || ((AF) == GPIO_AF_TIM14) || \
00337                                     ((AF) == GPIO_AF_USART1) || ((AF) == GPIO_AF_USART2) || \
00338                                     ((AF) == GPIO_AF_USART3) || ((AF) == GPIO_AF_UART4) || \
00339                                     ((AF) == GPIO_AF_UART5) || ((AF) == GPIO_AF_USART6) || \
00340                                     ((AF) == GPIO_AF_CAN1) || ((AF) == GPIO_AF_CAN2) || \
00341                                     ((AF) == GPIO_AF_OTG_FS) || ((AF) == GPIO_AF_OTG_HS) || \
00342                                     ((AF) == GPIO_AF_ETH) || ((AF) == GPIO_AF_FSMC) || \
00343                                     ((AF) == GPIO_AF_OTG_HS_FS) || ((AF) == GPIO_AF_SDIO) || \
00344                                     ((AF) == GPIO_AF_DCMCI) || ((AF) == GPIO_AF_EVENTOUT))
00353 #define GPIO_Mode_AIN                GPIO_Mode_AN
00354
00355 #define GPIO_AF_OTG1_FS              GPIO_AF_OTG_FS
00356 #define GPIO_AF_OTG2_HS              GPIO_AF_OTG_HS
00357 #define GPIO_AF_OTG2_FS              GPIO_AF_OTG_HS_FS
00358
00367 /* Exported macro -----*/
00368 /* Exported functions -----*/
00369
00370 /* Function used to set the GPIO configuration to the default reset state ****/
00371 void GPIO_DeInit(GPIO_TypeDef* GPIOx);
00372
00373 /* Initialization and Configuration functions -----*/
00374 void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);
00375 void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct);
00376 void GPIO_PinLockConfig(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
00377
00378 /* GPIO Read and Write functions -----*/
00379 uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
00380 uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
00381 uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
00382 uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
00383 void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
00384 void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
00385 void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
00386 void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);
00387 void GPIO_ToggleBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
00388
00389 /* GPIO Alternate functions configuration function -----*/
00390 void GPIO_PinAFConfig(GPIO_TypeDef* GPIOx, uint16_t GPIO_PinSource, uint8_t GPIO_AF);
00391
00392 #ifdef __cplusplus
00393 }
00394 #endif
00395
00396 #endif /*__STM32F4xx_GPIO_H */
00397
00406 /***** (C) COPYRIGHT 2011 STMicroelectronics *****/

```

6.21 drivers/stm32f4xx_i2c.c File Reference

This file provides firmware functions to manage the following functionalities of the Inter-integrated circuit (I2C)

```

#include "stm32f4xx_i2c.h"
#include "stm32f4xx_rcc.h"

```

Macros

- #define [CR1_CLEAR_MASK](#) ((uint16_t)0xFBF5) /*<! I2C registers Masks */
- #define [FLAG_MASK](#) ((uint32_t)0x00FFFFFF) /*<! I2C FLAG mask */
- #define [ITEN_MASK](#) ((uint32_t)0x07000000) /*<! I2C Interrupt Enable mask */

Functions

- void [I2C_DeInit](#) (I2C_TypeDef *I2Cx)
Deinitialize the I2Cx peripheral registers to their default reset values.
- void [I2C_Init](#) (I2C_TypeDef *I2Cx, [I2C_InitTypeDef](#) *I2C_InitStruct)
Initializes the I2Cx peripheral according to the specified parameters in the I2C_InitStruct.
- void [I2C_StructInit](#) ([I2C_InitTypeDef](#) *I2C_InitStruct)
Fills each I2C_InitStruct member with its default value.
- void [I2C_Cmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C peripheral.
- void [I2C_GenerateSTART](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Generates I2Cx communication START condition.
- void [I2C_GenerateSTOP](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Generates I2Cx communication STOP condition.
- void [I2C_Send7bitAddress](#) (I2C_TypeDef *I2Cx, uint8_t Address, uint8_t I2C_Direction)
Transmits the address byte to select the slave device.
- void [I2C_AcknowledgeConfig](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C acknowledge feature.
- void [I2C_OwnAddress2Config](#) (I2C_TypeDef *I2Cx, uint8_t Address)
Configures the specified I2C own address2.
- void [I2C_DualAddressCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C dual addressing mode.
- void [I2C_GeneralCallCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C general call feature.
- void [I2C_SoftwareResetCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C software reset.
- void [I2C_StretchClockCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C Clock stretching.
- void [I2C_FastModeDutyCycleConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_DutyCycle)
Selects the specified I2C fast mode duty cycle.
- void [I2C_NACKPositionConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_NACKPosition)
Selects the specified I2C NACK position in master receiver mode.
- void [I2C_SMBusAlertConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_SMBusAlert)
Drives the SMBusAlert pin high or low for the specified I2C.
- void [I2C_ARPCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C ARP.
- void [I2C_SendData](#) (I2C_TypeDef *I2Cx, uint8_t Data)
Sends a data byte through the I2Cx peripheral.
- uint8_t [I2C_ReceiveData](#) (I2C_TypeDef *I2Cx)
Returns the most recent received data by the I2Cx peripheral.
- void [I2C_TransmitPEC](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C PEC transfer.
- void [I2C_PECPositionConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_PECPosition)
Selects the specified I2C PEC position.
- void [I2C_CalculatePEC](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the PEC value calculation of the transferred bytes.
- uint8_t [I2C_GetPEC](#) (I2C_TypeDef *I2Cx)
Returns the PEC value for the specified I2C.
- void [I2C_DMAMCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C DMA requests.
- void [I2C_DMALastTransferCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)

- Specifies that the next DMA transfer is the last one.*
- uint16_t [I2C_ReadRegister](#) (I2C_TypeDef *I2Cx, uint8_t I2C_Register)
 - Reads the specified I2C register and returns its value.*
- void [I2C_ITConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_IT, FunctionalState NewState)
 - Enables or disables the specified I2C interrupts.*
- ErrorStatus [I2C_CheckEvent](#) (I2C_TypeDef *I2Cx, uint32_t I2C_EVENT)
 - Checks whether the last I2Cx Event is equal to the one passed as parameter.*
- uint32_t [I2C_GetLastEvent](#) (I2C_TypeDef *I2Cx)
 - Returns the last I2Cx Event.*
- FlagStatus [I2C_GetFlagStatus](#) (I2C_TypeDef *I2Cx, uint32_t I2C_FLAG)
 - Checks whether the specified I2C flag is set or not.*
- void [I2C_ClearFlag](#) (I2C_TypeDef *I2Cx, uint32_t I2C_FLAG)
 - Clears the I2Cx's pending flags.*
- ITStatus [I2C_GetITStatus](#) (I2C_TypeDef *I2Cx, uint32_t I2C_IT)
 - Checks whether the specified I2C interrupt has occurred or not.*
- void [I2C_ClearITPendingBit](#) (I2C_TypeDef *I2Cx, uint32_t I2C_IT)
 - Clears the I2Cx's interrupt pending bits.*

6.21.1 Detailed Description

This file provides firmware functions to manage the following functionalities of the Inter-integrated circuit (I2C)

Author

MCD Application Team

Version

V1.0.0

Date

30-September-2011

- Initialization and Configuration
- Data transfers
- PEC management
- DMA transfers management
- Interrupts, events and flags management

```
*
*  =====
*                                     How to use this driver
*  =====
*  1. Enable peripheral clock using RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2Cx, ENABLE)
*     function for I2C1, I2C2 or I2C3.
*
*  2. Enable SDA, SCL and SMBA (when used) GPIO clocks using
*     RCC_AHBPeriphClockCmd() function.
*
*  3. Peripheral alternate function:
*     - Connect the pin to the desired peripherals' Alternate
*       Function (AF) using GPIO_PinAFConfig() function
*     - Configure the desired pin in alternate function by:
*       GPIO_InitStruct->GPIO_Mode = GPIO_Mode_AF
*     - Select the type, pull-up/pull-down and output speed via
*       GPIO_PuPd, GPIO_OType and GPIO_Speed members
*     - Call GPIO_Init() function
*     Recommended configuration is Push-Pull, Pull-up, Open-Drain.
*     Add an external pull up if necessary (typically 4.7 KOhm).
```

```

*      4. Program the Mode, duty cycle , Own address, Ack, Speed and Acknowledged
*      Address using the I2C_Init() function.
*
*      5. Optionally you can enable/configure the following parameters without
*      re-initialization (i.e there is no need to call again I2C_Init() function):
*      - Enable the acknowledge feature using I2C_AcknowledgeConfig() function
*      - Enable the dual addressing mode using I2C_DualAddressCmd() function
*      - Enable the general call using the I2C_GeneralCallCmd() function
*      - Enable the clock stretching using I2C_StretchClockCmd() function
*      - Enable the fast mode duty cycle using the I2C_FastModeDutyCycleConfig()
*      function.
*      - Configure the NACK position for Master Receiver mode in case of
*      2 bytes reception using the function I2C_NACKPositionConfig().
*      - Enable the PEC Calculation using I2C_CalculatePEC() function
*      - For SMBus Mode:
*      - Enable the Address Resolution Protocol (ARP) using I2C_ARPCmd() function
*      - Configure the SMBusAlert pin using I2C_SMBusAlertConfig() function
*
*      6. Enable the NVIC and the corresponding interrupt using the function
*      I2C_ITConfig() if you need to use interrupt mode.
*
*      7. When using the DMA mode
*      - Configure the DMA using DMA_Init() function
*      - Active the needed channel Request using I2C_DMAMCmd() or
*      I2C_DMALastTransferCmd() function.
*      @note When using DMA mode, I2C interrupts may be used at the same time to
*      control the communication flow (Start/Stop/Ack... events and errors).
*
*      8. Enable the I2C using the I2C_Cmd() function.
*
*      9. Enable the DMA using the DMA_Cmd() function when using DMA mode in the
*      transfers.
*
*
*

```

Attention

THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

© COPYRIGHT 2011 STMicroelectronics

6.22 drivers/stm32f4xx_i2c.h File Reference

This file contains all the functions prototypes for the I2C firmware library.

```
#include "stm32f4xx.h"
```

Data Structures

- struct [I2C_InitTypeDef](#)
I2C Init structure definition

Macros

- #define [IS_I2C_ALL_PERIPH](#)(PERIPH)

- #define I2C_Mode_I2C ((uint16_t)0x0000)
- #define I2C_Mode_SMBusDevice ((uint16_t)0x0002)
- #define I2C_Mode_SMBusHost ((uint16_t)0x000A)
- #define IS_I2C_MODE(MODE)
- #define I2C_DutyCycle_16_9 ((uint16_t)0x4000)
- #define I2C_DutyCycle_2 ((uint16_t)0xBFFF)
- #define IS_I2C_DUTY_CYCLE(CYCLE)
- #define I2C_Ack_Enable ((uint16_t)0x0400)
- #define I2C_Ack_Disable ((uint16_t)0x0000)
- #define IS_I2C_ACK_STATE(STATE)
- #define I2C_Direction_Transmitter ((uint8_t)0x00)
- #define I2C_Direction_Receiver ((uint8_t)0x01)
- #define IS_I2C_DIRECTION(DIRECTION)
- #define I2C_AcknowledgedAddress_7bit ((uint16_t)0x4000)
- #define I2C_AcknowledgedAddress_10bit ((uint16_t)0xC000)
- #define IS_I2C_ACKNOWLEDGE_ADDRESS(ADDRESS)
- #define I2C_Register_CR1 ((uint8_t)0x00)
- #define I2C_Register_CR2 ((uint8_t)0x04)
- #define I2C_Register_OAR1 ((uint8_t)0x08)
- #define I2C_Register_OAR2 ((uint8_t)0x0C)
- #define I2C_Register_DR ((uint8_t)0x10)
- #define I2C_Register_SR1 ((uint8_t)0x14)
- #define I2C_Register_SR2 ((uint8_t)0x18)
- #define I2C_Register_CCR ((uint8_t)0x1C)
- #define I2C_Register_TRISE ((uint8_t)0x20)
- #define IS_I2C_REGISTER(REGISTER)
- #define I2C_NACKPosition_Next ((uint16_t)0x0800)
- #define I2C_NACKPosition_Current ((uint16_t)0xF7FF)
- #define IS_I2C_NACK_POSITION(POSITION)
- #define I2C_SMBusAlert_Low ((uint16_t)0x2000)
- #define I2C_SMBusAlert_High ((uint16_t)0xDFFF)
- #define IS_I2C_SMBUS_ALERT(ALERT)
- #define I2C_PECPosition_Next ((uint16_t)0x0800)
- #define I2C_PECPosition_Current ((uint16_t)0xF7FF)
- #define IS_I2C_PEC_POSITION(POSITION)
- #define I2C_IT_BUF ((uint16_t)0x0400)
- #define I2C_IT_EVT ((uint16_t)0x0200)
- #define I2C_IT_ERR ((uint16_t)0x0100)
- #define IS_I2C_CONFIG_IT(IT) (((IT) & (uint16_t)0xF8FF) == 0x00) && ((IT) != 0x00)
- #define I2C_IT_SMBALERT ((uint32_t)0x01008000)
- #define I2C_IT_TIMEOUT ((uint32_t)0x01004000)
- #define I2C_IT_PECERR ((uint32_t)0x01001000)
- #define I2C_IT_OVR ((uint32_t)0x01000800)
- #define I2C_IT_AF ((uint32_t)0x01000400)
- #define I2C_IT_ARLO ((uint32_t)0x01000200)
- #define I2C_IT_BERR ((uint32_t)0x01000100)
- #define I2C_IT_TXE ((uint32_t)0x06000080)
- #define I2C_IT_RXNE ((uint32_t)0x06000040)
- #define I2C_IT_STOPF ((uint32_t)0x02000010)
- #define I2C_IT_ADD10 ((uint32_t)0x02000008)
- #define I2C_IT_BTF ((uint32_t)0x02000004)
- #define I2C_IT_ADDR ((uint32_t)0x02000002)
- #define I2C_IT_SB ((uint32_t)0x02000001)
- #define IS_I2C_CLEAR_IT(IT) (((IT) & (uint16_t)0x20FF) == 0x00) && ((IT) != (uint16_t)0x00)
- #define IS_I2C_GET_IT(IT)

- #define `I2C_FLAG_DUALF` ((uint32_t)0x00800000)
SR2 register flags
- #define `I2C_FLAG_SMBHOST` ((uint32_t)0x00400000)
- #define `I2C_FLAG_SMBDEFAULT` ((uint32_t)0x00200000)
- #define `I2C_FLAG_GENCALL` ((uint32_t)0x00100000)
- #define `I2C_FLAG_TRA` ((uint32_t)0x00040000)
- #define `I2C_FLAG_BUSY` ((uint32_t)0x00020000)
- #define `I2C_FLAG_MSL` ((uint32_t)0x00010000)
- #define `I2C_FLAG_SMBALERT` ((uint32_t)0x10008000)
SR1 register flags
- #define `I2C_FLAG_TIMEOUT` ((uint32_t)0x10004000)
- #define `I2C_FLAG_PECERR` ((uint32_t)0x10001000)
- #define `I2C_FLAG_OVR` ((uint32_t)0x10000800)
- #define `I2C_FLAG_AF` ((uint32_t)0x10000400)
- #define `I2C_FLAG_ARLO` ((uint32_t)0x10000200)
- #define `I2C_FLAG_BERR` ((uint32_t)0x10000100)
- #define `I2C_FLAG_TXE` ((uint32_t)0x10000080)
- #define `I2C_FLAG_RXNE` ((uint32_t)0x10000040)
- #define `I2C_FLAG_STOPF` ((uint32_t)0x10000010)
- #define `I2C_FLAG_ADD10` ((uint32_t)0x10000008)
- #define `I2C_FLAG_BTF` ((uint32_t)0x10000004)
- #define `I2C_FLAG_ADDR` ((uint32_t)0x10000002)
- #define `I2C_FLAG_SB` ((uint32_t)0x10000001)
- #define `IS_I2C_CLEAR_FLAG(FLAG)` (((FLAG) & (uint16_t)0x20FF) == 0x00) && ((FLAG) != (uint16_t)0x00)
- #define `IS_I2C_GET_FLAG(FLAG)`
- #define `I2C_EVENT_MASTER_MODE_SELECT` ((uint32_t)0x00030001) /* BUSY, MSL and SB flag */
Communication start.
- #define `I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED` ((uint32_t)0x00070082) /* BUSY, MSL, ADDR, TXE and TRA flags */
Address Acknowledge.
- #define `I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED` ((uint32_t)0x00030002) /* BUSY, MSL and ADDR flags */
- #define `I2C_EVENT_MASTER_MODE_ADDRESS10` ((uint32_t)0x00030008) /* BUSY, MSL and ADD10 flags */
- #define `I2C_EVENT_MASTER_BYTE_RECEIVED` ((uint32_t)0x00030040) /* BUSY, MSL and RXNE flags */
Communication events.
- #define `I2C_EVENT_MASTER_BYTE_TRANSMITTING` ((uint32_t)0x00070080) /* TRA, BUSY, MSL, TXE flags */
- #define `I2C_EVENT_MASTER_BYTE_TRANSMITTED` ((uint32_t)0x00070084) /* TRA, BUSY, MSL, TXE and BTF flags */
- #define `I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED` ((uint32_t)0x00020002) /* BUSY and ADDR flags */
Communication start events.
- #define `I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED` ((uint32_t)0x00060082) /* TRA, BUSY, TXE and ADDR flags */
- #define `I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED` ((uint32_t)0x00820000) /* DUALF and BUSY flags */
- #define `I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED` ((uint32_t)0x00860080) /* DUALF, TRA, BUSY and TXE flags */
- #define `I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED` ((uint32_t)0x00120000) /* GENCALL and BUSY flags */

- `#define I2C_EVENT_SLAVE_BYTE_RECEIVED ((uint32_t)0x00020040) /* BUSY and RXNE flags */`
Communication events.
- `#define I2C_EVENT_SLAVE_STOP_DETECTED ((uint32_t)0x00000010) /* STOPF flag */`
- `#define I2C_EVENT_SLAVE_BYTE_TRANSMITTED ((uint32_t)0x00060084) /* TRA, BUSY, TXE and BTF flags */`
- `#define I2C_EVENT_SLAVE_BYTE_TRANSMITTING ((uint32_t)0x00060080) /* TRA, BUSY and TXE flags */`
- `#define I2C_EVENT_SLAVE_ACK_FAILURE ((uint32_t)0x00000400) /* AF flag */`
- `#define IS_I2C_EVENT(EVENT)`
- `#define IS_I2C_OWN_ADDRESS1(ADDRESS1) ((ADDRESS1) <= 0x3FF)`
- `#define IS_I2C_CLOCK_SPEED(SPEED) (((SPEED) >= 0x1) && ((SPEED) <= 400000))`

Functions

- void `I2C_DeInit` (I2C_TypeDef *I2Cx)
Deinitialize the I2Cx peripheral registers to their default reset values.
- void `I2C_Init` (I2C_TypeDef *I2Cx, I2C_InitTypeDef *I2C_InitStruct)
Initializes the I2Cx peripheral according to the specified parameters in the I2C_InitStruct.
- void `I2C_StructInit` (I2C_InitTypeDef *I2C_InitStruct)
Fills each I2C_InitStruct member with its default value.
- void `I2C_Cmd` (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C peripheral.
- void `I2C_GenerateSTART` (I2C_TypeDef *I2Cx, FunctionalState NewState)
Generates I2Cx communication START condition.
- void `I2C_GenerateSTOP` (I2C_TypeDef *I2Cx, FunctionalState NewState)
Generates I2Cx communication STOP condition.
- void `I2C_Send7bitAddress` (I2C_TypeDef *I2Cx, uint8_t Address, uint8_t I2C_Direction)
Transmits the address byte to select the slave device.
- void `I2C_AcknowledgeConfig` (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C acknowledge feature.
- void `I2C_OwnAddress2Config` (I2C_TypeDef *I2Cx, uint8_t Address)
Configures the specified I2C own address2.
- void `I2C_DualAddressCmd` (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C dual addressing mode.
- void `I2C_GeneralCallCmd` (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C general call feature.
- void `I2C_SoftwareResetCmd` (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C software reset.
- void `I2C_StretchClockCmd` (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C Clock stretching.
- void `I2C_FastModeDutyCycleConfig` (I2C_TypeDef *I2Cx, uint16_t I2C_DutyCycle)
Selects the specified I2C fast mode duty cycle.
- void `I2C_NACKPositionConfig` (I2C_TypeDef *I2Cx, uint16_t I2C_NACKPosition)
Selects the specified I2C NACK position in master receiver mode.
- void `I2C_SMBusAlertConfig` (I2C_TypeDef *I2Cx, uint16_t I2C_SMBusAlert)
Drives the SMBusAlert pin high or low for the specified I2C.
- void `I2C_ARPCmd` (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C ARP.
- void `I2C_SendData` (I2C_TypeDef *I2Cx, uint8_t Data)
Sends a data byte through the I2Cx peripheral.
- uint8_t `I2C_ReceiveData` (I2C_TypeDef *I2Cx)
Returns the most recent received data by the I2Cx peripheral.

- void [I2C_TransmitPEC](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C PEC transfer.
- void [I2C_PECPositionConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_PECPosition)
Selects the specified I2C PEC position.
- void [I2C_CalculatePEC](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the PEC value calculation of the transferred bytes.
- uint8_t [I2C_GetPEC](#) (I2C_TypeDef *I2Cx)
Returns the PEC value for the specified I2C.
- void [I2C_DMAMCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Enables or disables the specified I2C DMA requests.
- void [I2C_DMALastTransferCmd](#) (I2C_TypeDef *I2Cx, FunctionalState NewState)
Specifies that the next DMA transfer is the last one.
- uint16_t [I2C_ReadRegister](#) (I2C_TypeDef *I2Cx, uint8_t I2C_Register)
Reads the specified I2C register and returns its value.
- void [I2C_ITConfig](#) (I2C_TypeDef *I2Cx, uint16_t I2C_IT, FunctionalState NewState)
Enables or disables the specified I2C interrupts.
- ErrorStatus [I2C_CheckEvent](#) (I2C_TypeDef *I2Cx, uint32_t I2C_EVENT)
Checks whether the last I2Cx Event is equal to the one passed as parameter.
- uint32_t [I2C_GetLastEvent](#) (I2C_TypeDef *I2Cx)
Returns the last I2Cx Event.
- FlagStatus [I2C_GetFlagStatus](#) (I2C_TypeDef *I2Cx, uint32_t I2C_FLAG)
Checks whether the specified I2C flag is set or not.
- void [I2C_ClearFlag](#) (I2C_TypeDef *I2Cx, uint32_t I2C_FLAG)
Clears the I2Cx's pending flags.
- ITStatus [I2C_GetITStatus](#) (I2C_TypeDef *I2Cx, uint32_t I2C_IT)
Checks whether the specified I2C interrupt has occurred or not.
- void [I2C_ClearITPendingBit](#) (I2C_TypeDef *I2Cx, uint32_t I2C_IT)
Clears the I2Cx's interrupt pending bits.

6.22.1 Detailed Description

This file contains all the functions prototypes for the I2C firmware library.

Author

MCD Application Team

Version

V1.0.0

Date

30-September-2011

Attention

THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

© COPYRIGHT 2011 STMicroelectronics

6.23 stm32f4xx_i2c.h

[Go to the documentation of this file.](#)

```

00001
00023 /* Define to prevent recursive inclusion -----*/
00024 #ifndef __STM32F4xx_I2C_H
00025 #define __STM32F4xx_I2C_H
00026
00027 #ifdef __cplusplus
00028     extern "C" {
00029 #endif
00030
00031 /* Includes -----*/
00032 #include "stm32f4xx.h"
00033
00042 /* Exported types -----*/
00043
00048 typedef struct
00049 {
00050     uint32_t I2C_ClockSpeed;
00053     uint16_t I2C_Mode;
00056     uint16_t I2C_DutyCycle;
00059     uint16_t I2C_OwnAddress1;
00062     uint16_t I2C_Ack;
00065     uint16_t I2C_AcknowledgedAddress;
00067 }I2C_InitTypeDef;
00068
00069 /* Exported constants -----*/
00070
00071
00076 #define IS_I2C_ALL_PERIPH(PERIPH) (((PERIPH) == I2C1) || \
00077                                     ((PERIPH) == I2C2) || \
00078                                     ((PERIPH) == I2C3))
00083 #define I2C_Mode_I2C                ((uint16_t)0x0000)
00084 #define I2C_Mode_SMBusDevice        ((uint16_t)0x0002)
00085 #define I2C_Mode_SMBusHost          ((uint16_t)0x000A)
00086 #define IS_I2C_MODE(MODE) (((MODE) == I2C_Mode_I2C) || \
00087                             ((MODE) == I2C_Mode_SMBusDevice) || \
00088                             ((MODE) == I2C_Mode_SMBusHost))
00097 #define I2C_DutyCycle_16_9          ((uint16_t)0x4000)
00098 #define I2C_DutyCycle_2             ((uint16_t)0xBFFF)
00099 #define IS_I2C_DUTY_CYCLE(CYCLE) (((CYCLE) == I2C_DutyCycle_16_9) || \
00100                                   ((CYCLE) == I2C_DutyCycle_2))
00109 #define I2C_Ack_Enable              ((uint16_t)0x0400)
00110 #define I2C_Ack_Disable             ((uint16_t)0x0000)
00111 #define IS_I2C_ACK_STATE(STATE) (((STATE) == I2C_Ack_Enable) || \
00112                                   ((STATE) == I2C_Ack_Disable))
00121 #define I2C_Direction_Transmitter   ((uint8_t)0x00)
00122 #define I2C_Direction_Receiver      ((uint8_t)0x01)
00123 #define IS_I2C_DIRECTION(DIRECTION) (((DIRECTION) == I2C_Direction_Transmitter) || \
00124                                       ((DIRECTION) == I2C_Direction_Receiver))
00133 #define I2C_AcknowledgedAddress_7bit ((uint16_t)0x4000)
00134 #define I2C_AcknowledgedAddress_10bit ((uint16_t)0xC000)
00135 #define IS_I2C_ACKNOWLEDGE_ADDRESS(ADDRESS) (((ADDRESS) == I2C_AcknowledgedAddress_7bit) || \
00136                                               ((ADDRESS) == I2C_AcknowledgedAddress_10bit))
00145 #define I2C_Register_CR1            ((uint8_t)0x00)
00146 #define I2C_Register_CR2            ((uint8_t)0x04)
00147 #define I2C_Register_OAR1          ((uint8_t)0x08)
00148 #define I2C_Register_OAR2          ((uint8_t)0x0C)
00149 #define I2C_Register_DR             ((uint8_t)0x10)
00150 #define I2C_Register_SR1            ((uint8_t)0x14)
00151 #define I2C_Register_SR2            ((uint8_t)0x18)
00152 #define I2C_Register_CCR            ((uint8_t)0x1C)
00153 #define I2C_Register_TRISE          ((uint8_t)0x20)
00154 #define IS_I2C_REGISTER(REGISTER) (((REGISTER) == I2C_Register_CR1) || \
00155                                     ((REGISTER) == I2C_Register_CR2) || \
00156                                     ((REGISTER) == I2C_Register_OAR1) || \
00157                                     ((REGISTER) == I2C_Register_OAR2) || \
00158                                     ((REGISTER) == I2C_Register_DR) || \
00159                                     ((REGISTER) == I2C_Register_SR1) || \
00160                                     ((REGISTER) == I2C_Register_SR2) || \
00161                                     ((REGISTER) == I2C_Register_CCR) || \
00162                                     ((REGISTER) == I2C_Register_TRISE))
00171 #define I2C_NACKPosition_Next       ((uint16_t)0x0800)
00172 #define I2C_NACKPosition_Current    ((uint16_t)0xF7FF)
00173 #define IS_I2C_NACK_POSITION(POSITION) (((POSITION) == I2C_NACKPosition_Next) || \
00174                                           ((POSITION) == I2C_NACKPosition_Current))
00183 #define I2C_SMBusAlert_Low          ((uint16_t)0x2000)
00184 #define I2C_SMBusAlert_High         ((uint16_t)0xDFFF)
00185 #define IS_I2C_SMBUS_ALERT(ALERT) (((ALERT) == I2C_SMBusAlert_Low) || \

```

```

00186                                     ((ALERT) == I2C_SMBusAlert_High))
00195 #define I2C_PECPosition_Next          ((uint16_t)0x0800)
00196 #define I2C_PECPosition_Current        ((uint16_t)0xF7FF)
00197 #define IS_I2C_PEC_POSITION(POSITION) (((POSITION) == I2C_PECPosition_Next) || \
00198                                     ((POSITION) == I2C_PECPosition_Current))
00207 #define I2C_IT_BUF                     ((uint16_t)0x0400)
00208 #define I2C_IT_EVT                     ((uint16_t)0x0200)
00209 #define I2C_IT_ERR                     ((uint16_t)0x0100)
00210 #define IS_I2C_CONFIG_IT(IT) (((IT) & (uint16_t)0xF8FF) == 0x00) && ((IT) != 0x00))
00219 #define I2C_IT_SMBALERT                 ((uint32_t)0x01008000)
00220 #define I2C_IT_TIMEOUT                 ((uint32_t)0x01004000)
00221 #define I2C_IT_PECERR                 ((uint32_t)0x01001000)
00222 #define I2C_IT_OVR                     ((uint32_t)0x01000800)
00223 #define I2C_IT_AF                     ((uint32_t)0x01000400)
00224 #define I2C_IT_ARLO                     ((uint32_t)0x01000200)
00225 #define I2C_IT_BERR                     ((uint32_t)0x01000100)
00226 #define I2C_IT_TXE                     ((uint32_t)0x06000080)
00227 #define I2C_IT_RXNE                     ((uint32_t)0x06000040)
00228 #define I2C_IT_STOPF                   ((uint32_t)0x02000010)
00229 #define I2C_IT_ADD10                   ((uint32_t)0x02000008)
00230 #define I2C_IT_BTF                     ((uint32_t)0x02000004)
00231 #define I2C_IT_ADDR                     ((uint32_t)0x02000002)
00232 #define I2C_IT_SB                      ((uint32_t)0x02000001)
00233
00234 #define IS_I2C_CLEAR_IT(IT) (((IT) & (uint16_t)0x20FF) == 0x00) && ((IT) != (uint16_t)0x00))
00235
00236 #define IS_I2C_GET_IT(IT) (((IT) == I2C_IT_SMBALERT) || ((IT) == I2C_IT_TIMEOUT) || \
00237                             ((IT) == I2C_IT_PECERR) || ((IT) == I2C_IT_OVR) || \
00238                             ((IT) == I2C_IT_AF) || ((IT) == I2C_IT_ARLO) || \
00239                             ((IT) == I2C_IT_BERR) || ((IT) == I2C_IT_TXE) || \
00240                             ((IT) == I2C_IT_RXNE) || ((IT) == I2C_IT_STOPF) || \
00241                             ((IT) == I2C_IT_ADD10) || ((IT) == I2C_IT_BTF) || \
00242                             ((IT) == I2C_IT_ADDR) || ((IT) == I2C_IT_SB))
00255 #define I2C_FLAG_DUALF                 ((uint32_t)0x00800000)
00256 #define I2C_FLAG_SMBHOST               ((uint32_t)0x00400000)
00257 #define I2C_FLAG_SMBDEFAULT            ((uint32_t)0x00200000)
00258 #define I2C_FLAG_GENCALL               ((uint32_t)0x00100000)
00259 #define I2C_FLAG_TRA                   ((uint32_t)0x00040000)
00260 #define I2C_FLAG_BUSY                  ((uint32_t)0x00020000)
00261 #define I2C_FLAG_MSL                   ((uint32_t)0x00010000)
00262
00267 #define I2C_FLAG_SMBALERT              ((uint32_t)0x10008000)
00268 #define I2C_FLAG_TIMEOUT               ((uint32_t)0x10004000)
00269 #define I2C_FLAG_PECERR                ((uint32_t)0x10001000)
00270 #define I2C_FLAG_OVR                  ((uint32_t)0x10000800)
00271 #define I2C_FLAG_AF                    ((uint32_t)0x10000400)
00272 #define I2C_FLAG_ARLO                  ((uint32_t)0x10000200)
00273 #define I2C_FLAG_BERR                  ((uint32_t)0x10000100)
00274 #define I2C_FLAG_TXE                   ((uint32_t)0x10000080)
00275 #define I2C_FLAG_RXNE                  ((uint32_t)0x10000040)
00276 #define I2C_FLAG_STOPF                 ((uint32_t)0x10000010)
00277 #define I2C_FLAG_ADD10                 ((uint32_t)0x10000008)
00278 #define I2C_FLAG_BTF                   ((uint32_t)0x10000004)
00279 #define I2C_FLAG_ADDR                  ((uint32_t)0x10000002)
00280 #define I2C_FLAG_SB                    ((uint32_t)0x10000001)
00281
00282 #define IS_I2C_CLEAR_FLAG(FLAG) (((FLAG) & (uint16_t)0x20FF) == 0x00) && ((FLAG) != (uint16_t)0x00))
00283
00284 #define IS_I2C_GET_FLAG(FLAG) (((FLAG) == I2C_FLAG_DUALF) || ((FLAG) == I2C_FLAG_SMBHOST) || \
00285                                 ((FLAG) == I2C_FLAG_SMBDEFAULT) || ((FLAG) == I2C_FLAG_GENCALL) || \
00286                                 ((FLAG) == I2C_FLAG_TRA) || ((FLAG) == I2C_FLAG_BUSY) || \
00287                                 ((FLAG) == I2C_FLAG_MSL) || ((FLAG) == I2C_FLAG_SMBALERT) || \
00288                                 ((FLAG) == I2C_FLAG_TIMEOUT) || ((FLAG) == I2C_FLAG_PECERR) || \
00289                                 ((FLAG) == I2C_FLAG_OVR) || ((FLAG) == I2C_FLAG_AF) || \
00290                                 ((FLAG) == I2C_FLAG_ARLO) || ((FLAG) == I2C_FLAG_BERR) || \
00291                                 ((FLAG) == I2C_FLAG_TXE) || ((FLAG) == I2C_FLAG_RXNE) || \
00292                                 ((FLAG) == I2C_FLAG_STOPF) || ((FLAG) == I2C_FLAG_ADD10) || \
00293                                 ((FLAG) == I2C_FLAG_BTF) || ((FLAG) == I2C_FLAG_ADDR) || \
00294                                 ((FLAG) == I2C_FLAG_SB))
00317 /* --EV5 */
00318 #define I2C_EVENT_MASTER_MODE_SELECT    ((uint32_t)0x00030001) /* BUSY, MSL and SB
flag */
00319
00345 /* --EV6 */
00346 #define I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED ((uint32_t)0x00070082) /* BUSY, MSL, ADDR,
TXE and TRA flags */
00347 #define I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED ((uint32_t)0x00030002) /* BUSY, MSL and
ADDR flags */
00348 /* --EV9 */
00349 #define I2C_EVENT_MASTER_MODE_ADDRESS10 ((uint32_t)0x00030008) /* BUSY, MSL and
ADD10 flags */
00350
00380 /* Master RECEIVER mode -----*/
00381 /* --EV7 */
00382 #define I2C_EVENT_MASTER_BYTE_RECEIVED ((uint32_t)0x00030040) /* BUSY, MSL and
RXNE flags */
00383

```

```

00384 /* Master TRANSMITTER mode -----*/
00385 /* --EV8 */
00386 #define I2C_EVENT_MASTER_BYTE_TRANSMITTING ((uint32_t)0x00070080) /* TRA, BUSY, MSL,
TXE flags */
00387 /* --EV8_2 */
00388 #define I2C_EVENT_MASTER_BYTE_TRANSMITTED ((uint32_t)0x00070084) /* TRA, BUSY, MSL,
TXE and BTF flags */
00389
00390
00423 /* --EV1 (all the events below are variants of EV1) */
00424 /* 1) Case of One Single Address managed by the slave */
00425 #define I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED ((uint32_t)0x00020002) /* BUSY and ADDR
flags */
00426 #define I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED ((uint32_t)0x00060082) /* TRA, BUSY, TXE
and ADDR flags */
00427
00428 /* 2) Case of Dual address managed by the slave */
00429 #define I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED ((uint32_t)0x00820000) /* DUALF and BUSY
flags */
00430 #define I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED ((uint32_t)0x00860080) /* DUALF, TRA, BUSY
and TXE flags */
00431
00432 /* 3) Case of General Call enabled for the slave */
00433 #define I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED ((uint32_t)0x00120000) /* GENCALL and BUSY
flags */
00434
00462 /* Slave RECEIVER mode -----*/
00463 /* --EV2 */
00464 #define I2C_EVENT_SLAVE_BYTE_RECEIVED ((uint32_t)0x00020040) /* BUSY and RXNE
flags */
00465 /* --EV4 */
00466 #define I2C_EVENT_SLAVE_STOP_DETECTED ((uint32_t)0x00000010) /* STOPF flag */
00467
00468 /* Slave TRANSMITTER mode -----*/
00469 /* --EV3 */
00470 #define I2C_EVENT_SLAVE_BYTE_TRANSMITTED ((uint32_t)0x00060084) /* TRA, BUSY, TXE
and BTF flags */
00471 #define I2C_EVENT_SLAVE_BYTE_TRANSMITTING ((uint32_t)0x00060080) /* TRA, BUSY and
TXE flags */
00472 /* --EV3_2 */
00473 #define I2C_EVENT_SLAVE_ACK_FAILURE ((uint32_t)0x00000400) /* AF flag */
00474
00475 /*
00476 =====
00477 End of Events Description
00478 =====
00479 */
00480
00481 #define IS_I2C_EVENT(EVENT) (((EVENT) == I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED) || \
00482 (EVENT) == I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED) || \
00483 (EVENT) == I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED) || \
00484 (EVENT) == I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED) || \
00485 (EVENT) == I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED) || \
00486 (EVENT) == I2C_EVENT_SLAVE_BYTE_RECEIVED) || \
00487 (EVENT) == (I2C_EVENT_SLAVE_BYTE_RECEIVED | I2C_FLAG_DUALF)) || \
00488 (EVENT) == (I2C_EVENT_SLAVE_BYTE_RECEIVED | I2C_FLAG_GENCALL)) || \
00489 (EVENT) == I2C_EVENT_SLAVE_BYTE_TRANSMITTED) || \
00490 (EVENT) == (I2C_EVENT_SLAVE_BYTE_TRANSMITTED | I2C_FLAG_DUALF)) || \
00491 (EVENT) == (I2C_EVENT_SLAVE_BYTE_TRANSMITTED | I2C_FLAG_GENCALL)) || \
00492 (EVENT) == I2C_EVENT_SLAVE_STOP_DETECTED) || \
00493 (EVENT) == I2C_EVENT_MASTER_MODE_SELECT) || \
00494 (EVENT) == I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED) || \
00495 (EVENT) == I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED) || \
00496 (EVENT) == I2C_EVENT_MASTER_BYTE_RECEIVED) || \
00497 (EVENT) == I2C_EVENT_MASTER_BYTE_TRANSMITTED) || \
00498 (EVENT) == I2C_EVENT_MASTER_BYTE_TRANSMITTING) || \
00499 (EVENT) == I2C_EVENT_MASTER_MODE_ADDRESS10) || \
00500 (EVENT) == I2C_EVENT_SLAVE_ACK_FAILURE))
00509 #define IS_I2C_OWN_ADDRESS1(ADDRESS1) ((ADDRESS1) <= 0x3FF)
00518 #define IS_I2C_CLOCK_SPEED(SPEED) (((SPEED) >= 0x1) && ((SPEED) <= 400000))
00527 /* Exported macro -----*/
00528 /* Exported functions -----*/
00529
00530 /* Function used to set the I2C configuration to the default reset state *****/
00531 void I2C_DeInit(I2C_TypeDef* I2Cx);
00532
00533 /* Initialization and Configuration functions *****/
00534 void I2C_Init(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct);
00535 void I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct);
00536 void I2C_Cmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
00537 void I2C_GenerateSTART(I2C_TypeDef* I2Cx, FunctionalState NewState);
00538 void I2C_GenerateSTOP(I2C_TypeDef* I2Cx, FunctionalState NewState);
00539 void I2C_Send7bitAddress(I2C_TypeDef* I2Cx, uint8_t Address, uint8_t I2C_Direction);
00540 void I2C_AcknowledgeConfig(I2C_TypeDef* I2Cx, FunctionalState NewState);
00541 void I2C_OwnAddress2Config(I2C_TypeDef* I2Cx, uint8_t Address);
00542 void I2C_DualAddressCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
00543 void I2C_GeneralCallCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);

```

```

00544 void I2C_SoftwareResetCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
00545 void I2C_StretchClockCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
00546 void I2C_FastModeDutyCycleConfig(I2C_TypeDef* I2Cx, uint16_t I2C_DutyCycle);
00547 void I2C_NACKPositionConfig(I2C_TypeDef* I2Cx, uint16_t I2C_NACKPosition);
00548 void I2C_SMBusAlertConfig(I2C_TypeDef* I2Cx, uint16_t I2C_SMBusAlert);
00549 void I2C_ARPCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
00550
00551 /* Data transfers functions *****/
00552 void I2C_SendData(I2C_TypeDef* I2Cx, uint8_t Data);
00553 uint8_t I2C_ReceiveData(I2C_TypeDef* I2Cx);
00554
00555 /* PEC management functions *****/
00556 void I2C_TransmitPEC(I2C_TypeDef* I2Cx, FunctionalState NewState);
00557 void I2C_PECPositionConfig(I2C_TypeDef* I2Cx, uint16_t I2C_PECPosition);
00558 void I2C_CalculatePEC(I2C_TypeDef* I2Cx, FunctionalState NewState);
00559 uint8_t I2C_GetPEC(I2C_TypeDef* I2Cx);
00560
00561 /* DMA transfers management functions *****/
00562 void I2C_DMAMCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
00563 void I2C_DMALastTransferCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
00564
00565 /* Interrupts, events and flags management functions *****/
00566 uint16_t I2C_ReadRegister(I2C_TypeDef* I2Cx, uint8_t I2C_Register);
00567 void I2C_ITConfig(I2C_TypeDef* I2Cx, uint16_t I2C_IT, FunctionalState NewState);
00568
00569 /*
00570 =====
00571                      I2C State Monitoring Functions
00572 =====
00573 This I2C driver provides three different ways for I2C state monitoring
00574 depending on the application requirements and constraints:
00575
00576 1. Basic state monitoring (Using I2C_CheckEvent() function)
00577 -----
00578 It compares the status registers (SR1 and SR2) content to a given event
00579 (can be the combination of one or more flags).
00580 It returns SUCCESS if the current status includes the given flags
00581 and returns ERROR if one or more flags are missing in the current status.
00582
00583 - When to use
00584   - This function is suitable for most applications as well as for startup
00585     activity since the events are fully described in the product reference
00586     manual (RM0090).
00587   - It is also suitable for users who need to define their own events.
00588
00589 - Limitations
00590   - If an error occurs (ie. error flags are set besides to the monitored
00591     flags), the I2C_CheckEvent() function may return SUCCESS despite
00592     the communication hold or corrupted real state.
00593     In this case, it is advised to use error interrupts to monitor
00594     the error events and handle them in the interrupt IRQ handler.
00595
00596 Note
00597 For error management, it is advised to use the following functions:
00598   - I2C_ITConfig() to configure and enable the error interrupts (I2C_IT_ERR).
00599   - I2Cx_ER_IRQHandler() which is called when the error interrupt occurs.
00600     Where x is the peripheral instance (I2C1, I2C2 ...)
00601   - I2C_GetFlagStatus() or I2C_GetITStatus() to be called into the
00602     I2Cx_ER_IRQHandler() function in order to determine which error occurred.
00603   - I2C_ClearFlag() or I2C_ClearITPendingBit() and/or I2C_SoftwareResetCmd()
00604     and/or I2C_GenerateStop() in order to clear the error flag and source
00605     and return to correct communication status.
00606
00607
00608
00609 2. Advanced state monitoring (Using the function I2C_GetLastEvent())
00610 -----
00611 Using the function I2C_GetLastEvent() which returns the image of both status
00612 registers in a single word (uint32_t) (Status Register 2 value is shifted left
00613 by 16 bits and concatenated to Status Register 1).
00614
00615 - When to use
00616   - This function is suitable for the same applications above but it
00617     allows to overcome the mentioned limitation of I2C_GetFlagStatus()
00618     function.
00619   - The returned value could be compared to events already defined in
00620     this file or to custom values defined by user.
00621     This function is suitable when multiple flags are monitored at the
00622     same time.
00623   - At the opposite of I2C_CheckEvent() function, this function allows
00624     user to choose when an event is accepted (when all events flags are
00625     set and no other flags are set or just when the needed flags are set
00626     like I2C_CheckEvent() function.
00627
00628 - Limitations
00629   - User may need to define his own events.
00630   - Same remark concerning the error management is applicable for this

```

```

00631         function if user decides to check only regular communication flags
00632         (and ignores error flags).
00633
00634
00635     3. Flag-based state monitoring (Using the function I2C_GetFlagStatus())
00636     -----
00637
00638     Using the function I2C_GetFlagStatus() which simply returns the status of
00639     one single flag (ie. I2C_FLAG_RXNE ...).
00640
00641     - When to use
00642     - This function could be used for specific applications or in debug
00643       phase.
00644     - It is suitable when only one flag checking is needed (most I2C
00645       events are monitored through multiple flags).
00646     - Limitations:
00647     - When calling this function, the Status register is accessed.
00648       Some flags are cleared when the status register is accessed.
00649       So checking the status of one Flag, may clear other ones.
00650     - Function may need to be called twice or more in order to monitor
00651       one single event.
00652 */
00653
00654 /*
00655 =====
00656                               1. Basic state monitoring
00657 =====
00658 */
00659 ErrorStatus I2C_CheckEvent(I2C_TypeDef* I2Cx, uint32_t I2C_EVENT);
00660 /*
00661 =====
00662                               2. Advanced state monitoring
00663 =====
00664 */
00665 uint32_t I2C_GetLastEvent(I2C_TypeDef* I2Cx);
00666 /*
00667 =====
00668                               3. Flag-based state monitoring
00669 =====
00670 */
00671 FlagStatus I2C_GetFlagStatus(I2C_TypeDef* I2Cx, uint32_t I2C_FLAG);
00672
00673
00674 void I2C_ClearFlag(I2C_TypeDef* I2Cx, uint32_t I2C_FLAG);
00675 ITStatus I2C_GetITStatus(I2C_TypeDef* I2Cx, uint32_t I2C_IT);
00676 void I2C_ClearITPendingBit(I2C_TypeDef* I2Cx, uint32_t I2C_IT);
00677
00678 #ifdef __cplusplus
00679 }
00680 #endif
00681
00682 #endif /* __STM32F4xx_I2C_H */
00683
00692 /***** (C) COPYRIGHT 2011 STMicroelectronics *****/

```

6.24 drivers/stm32f4xx_rcc.c File Reference

This file provides firmware functions to manage the following functionalities of the Reset and clock control (RCC) peripheral:

```
#include "stm32f4xx_rcc.h"
```

Macros

- #define [HSE_VALUE](#) ((uint32_t)25000000)
 - #define [HSE_STARTUP_TIMEOUT](#) ((uint16_t)0x0500)
- In the following line adjust the External High Speed oscillator (HSE) Startup Timeout value.*
- #define [HSI_VALUE](#) ((uint32_t)16000000)
 - #define [RCC_OFFSET](#) (RCC_BASE - PERIPH_BASE)
 - #define [CR_OFFSET](#) ([RCC_OFFSET](#) + 0x00)
 - #define [HSION_BitNumber](#) 0x00
 - #define [CR_HSION_BB](#) (PERIPH_BB_BASE + ([CR_OFFSET](#) * 32) + ([HSION_BitNumber](#) * 4))
 - #define [CSSON_BitNumber](#) 0x13
 - #define [CR_CSSON_BB](#) (PERIPH_BB_BASE + ([CR_OFFSET](#) * 32) + ([CSSON_BitNumber](#) * 4))
 - #define [PLLON_BitNumber](#) 0x18

- #define `CR_PLLON_BB` (PERIPH_BB_BASE + (CR_OFFSET * 32) + (PLLON_BitNumber * 4))
- #define `PLLI2SON_BitNumber` 0x1A
- #define `CR_PLLI2SON_BB` (PERIPH_BB_BASE + (CR_OFFSET * 32) + (PLLI2SON_BitNumber * 4))
- #define `CFGR_OFFSET` (RCC_OFFSET + 0x08)
- #define `I2SSRC_BitNumber` 0x17
- #define `CFGR_I2SSRC_BB` (PERIPH_BB_BASE + (CFGR_OFFSET * 32) + (I2SSRC_BitNumber * 4))
- #define `BDCR_OFFSET` (RCC_OFFSET + 0x70)
- #define `RTCEN_BitNumber` 0x0F
- #define `BDCR_RTCEN_BB` (PERIPH_BB_BASE + (BDCR_OFFSET * 32) + (RTCEN_BitNumber * 4))
- #define `BDRST_BitNumber` 0x10
- #define `BDCR_BDRST_BB` (PERIPH_BB_BASE + (BDCR_OFFSET * 32) + (BDRST_BitNumber * 4))
- #define `CSR_OFFSET` (RCC_OFFSET + 0x74)
- #define `LSION_BitNumber` 0x00
- #define `CSR_LSION_BB` (PERIPH_BB_BASE + (CSR_OFFSET * 32) + (LSION_BitNumber * 4))
- #define `CFGR_MCO2_RESET_MASK` ((uint32_t)0x07FFFFFF)
- #define `CFGR_MCO1_RESET_MASK` ((uint32_t)0xF89FFFFFF)
- #define `FLAG_MASK` ((uint8_t)0x1F)
- #define `CR_BYTE3_ADDRESS` ((uint32_t)0x40023802)
- #define `CIR_BYTE2_ADDRESS` ((uint32_t)(RCC_BASE + 0x0C + 0x01))
- #define `CIR_BYTE3_ADDRESS` ((uint32_t)(RCC_BASE + 0x0C + 0x02))
- #define `BDCR_ADDRESS` (PERIPH_BASE + BDCR_OFFSET)

Functions

- void `RCC_DeInit` (void)
Resets the RCC clock configuration to the default reset state.
- void `RCC_HSEConfig` (uint8_t RCC_HSE)
Configures the External High Speed oscillator (HSE).
- ErrorStatus `RCC_WaitForHSEStartUp` (void)
Waits for HSE start-up.
- void `RCC_AdjustHSICalibrationValue` (uint8_t HSICalibrationValue)
Adjusts the Internal High Speed oscillator (HSI) calibration value.
- void `RCC_HSICmd` (FunctionalState NewState)
Enables or disables the Internal High Speed oscillator (HSI).
- void `RCC_LSEConfig` (uint8_t RCC_LSE)
Configures the External Low Speed oscillator (LSE).
- void `RCC_LSICmd` (FunctionalState NewState)
Enables or disables the Internal Low Speed oscillator (LSI).
- void `RCC_PLLConfig` (uint32_t RCC_PLLSource, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP, uint32_t PLLQ)
Configures the main PLL clock source, multiplication and division factors.
- void `RCC_PLLCmd` (FunctionalState NewState)
Enables or disables the main PLL.
- void `RCC_PLLI2SConfig` (uint32_t PLLI2SN, uint32_t PLLI2SR)
Configures the PLLI2S clock multiplication and division factors.
- void `RCC_PLLI2SCmd` (FunctionalState NewState)
Enables or disables the PLLI2S.
- void `RCC_ClockSecuritySystemCmd` (FunctionalState NewState)
Enables or disables the Clock Security System.
- void `RCC_MCO1Config` (uint32_t RCC_MCO1Source, uint32_t RCC_MCO1Div)
Selects the clock source to output on MCO1 pin(PA8).
- void `RCC_MCO2Config` (uint32_t RCC_MCO2Source, uint32_t RCC_MCO2Div)
Selects the clock source to output on MCO2 pin(PC9).

- void [RCC_SYSClkConfig](#) (uint32_t RCC_SYSClkSource)
Configures the system clock (SYSClk).
- uint8_t [RCC_GetSYSClkSource](#) (void)
Returns the clock source used as system clock.
- void [RCC_HCLKConfig](#) (uint32_t RCC_SYSClk)
Configures the AHB clock (HCLK).
- void [RCC_PCLK1Config](#) (uint32_t RCC_HCLK)
Configures the Low Speed APB clock (PCLK1).
- void [RCC_PCLK2Config](#) (uint32_t RCC_HCLK)
Configures the High Speed APB clock (PCLK2).
- void [RCC_GetClocksFreq](#) (RCC_ClocksTypeDef *RCC_Clocks)
Returns the frequencies of different on chip clocks; SYSClk, HCLK, PCLK1 and PCLK2.

- void [RCC_RTCCLKConfig](#) (uint32_t RCC_RTCCLKSource)
Configures the RTC clock (RTCCLK).
- void [RCC_RTCCLKCmd](#) (FunctionalState NewState)
Enables or disables the RTC clock.
- void [RCC_BackupResetCmd](#) (FunctionalState NewState)
Forces or releases the Backup domain reset.
- void [RCC_I2SCLKConfig](#) (uint32_t RCC_I2SCLKSource)
Configures the I2S clock source (I2SCLK).
- void [RCC_AHB1PeriphClockCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
Enables or disables the AHB1 peripheral clock.
- void [RCC_AHB2PeriphClockCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
Enables or disables the AHB2 peripheral clock.
- void [RCC_AHB3PeriphClockCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Enables or disables the AHB3 peripheral clock.
- void [RCC_APB1PeriphClockCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Enables or disables the Low Speed APB (APB1) peripheral clock.
- void [RCC_APB2PeriphClockCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Enables or disables the High Speed APB (APB2) peripheral clock.
- void [RCC_AHB1PeriphResetCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
Forces or releases AHB1 peripheral reset.
- void [RCC_AHB2PeriphResetCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
Forces or releases AHB2 peripheral reset.
- void [RCC_AHB3PeriphResetCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Forces or releases AHB3 peripheral reset.
- void [RCC_APB1PeriphResetCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Forces or releases Low Speed APB (APB1) peripheral reset.
- void [RCC_APB2PeriphResetCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Forces or releases High Speed APB (APB2) peripheral reset.
- void [RCC_AHB1PeriphClockLPModeCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
Enables or disables the AHB1 peripheral clock during Low Power (Sleep) mode.
- void [RCC_AHB2PeriphClockLPModeCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
Enables or disables the AHB2 peripheral clock during Low Power (Sleep) mode.
- void [RCC_AHB3PeriphClockLPModeCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Enables or disables the AHB3 peripheral clock during Low Power (Sleep) mode.
- void [RCC_APB1PeriphClockLPModeCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Enables or disables the APB1 peripheral clock during Low Power (Sleep) mode.
- void [RCC_APB2PeriphClockLPModeCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Enables or disables the APB2 peripheral clock during Low Power (Sleep) mode.

- void [RCC_ITConfig](#) (uint8_t RCC_IT, FunctionalState NewState)
Enables or disables the specified RCC interrupts.
- FlagStatus [RCC_GetFlagStatus](#) (uint8_t RCC_FLAG)
Checks whether the specified RCC flag is set or not.
- void [RCC_ClearFlag](#) (void)
Clears the RCC reset flags. The reset flags are: RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST, RCC_FLAG_LPWRST.
- ITStatus [RCC_GetITStatus](#) (uint8_t RCC_IT)
Checks whether the specified RCC interrupt has occurred or not.
- void [RCC_ClearITPendingBit](#) (uint8_t RCC_IT)
Clears the RCC's interrupt pending bits.

6.24.1 Detailed Description

This file provides firmware functions to manage the following functionalities of the Reset and clock control (RCC) peripheral:

Author

MCD Application Team

Version

V1.0.0

Date

30-September-2011

- Internal/external clocks, PLL, CSS and MCO configuration
- System, AHB and APB busses clocks configuration
- Peripheral clocks configuration
- Interrupts and flags management

```
*
*  =====
*                      RCC specific features
*  =====
*
*  After reset the device is running from Internal High Speed oscillator
*  (HSI 16MHz) with Flash 0 wait state, Flash prefetch buffer, D-Cache
*  and I-Cache are disabled, and all peripherals are off except internal
*  SRAM, Flash and JTAG.
*
*  - There is no prescaler on High speed (AHB) and Low speed (APB) busses;
*    all peripherals mapped on these busses are running at HSI speed.
*  - The clock for all peripherals is switched off, except the SRAM and FLASH.
*  - All GPIOs are in input floating state, except the JTAG pins which
*    are assigned to be used for debug purpose.
*
*  Once the device started from reset, the user application has to:
*  - Configure the clock source to be used to drive the System clock
*    (if the application needs higher frequency/performance)
*  - Configure the System clock frequency and Flash settings
*  - Configure the AHB and APB busses prescalers
*  - Enable the clock for the peripheral(s) to be used
*  - Configure the clock source(s) for peripherals which clocks are not
*    derived from the System clock (I2S, RTC, ADC, USB OTG FS/SDIO/RNG)
*
*
```

Attention

THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

© COPYRIGHT 2011 STMicroelectronics

6.24.2 Macro Definition Documentation

6.24.2.1 HSE_STARTUP_TIMEOUT

```
#define HSE_STARTUP_TIMEOUT ((uint16_t)0x0500)
```

In the following line adjust the External High Speed oscillator (HSE) Startup Timeout value.
Time out for HSE start up

6.24.2.2 HSE_VALUE

```
#define HSE_VALUE ((uint32_t)25000000)
```

Value of the External oscillator in Hz

6.24.2.3 HSI_VALUE

```
#define HSI_VALUE ((uint32_t)16000000)
```

Value of the Internal oscillator in Hz

6.25 drivers/stm32f4xx_rcc.h File Reference

This file contains all the functions prototypes for the RCC firmware library.

```
#include "stm32f4xx.h"
```

Data Structures

- struct [RCC_ClocksTypeDef](#)

Macros

- #define [RCC_HSE_OFF](#) ((uint8_t)0x00)
- #define [RCC_HSE_ON](#) ((uint8_t)0x01)
- #define [RCC_HSE_Bypass](#) ((uint8_t)0x05)
- #define [IS_RCC_HSE](#)(HSE)
- #define [RCC_PLLSource_HSI](#) ((uint32_t)0x00000000)
- #define [RCC_PLLSource_HSE](#) ((uint32_t)0x00400000)
- #define [IS_RCC_PLL_SOURCE](#)(SOURCE)
- #define [IS_RCC_PLLM_VALUE](#)(VALUE) ((VALUE) <= 63)
- #define [IS_RCC_PLLN_VALUE](#)(VALUE) ((192 <= (VALUE)) && ((VALUE) <= 432))
- #define [IS_RCC_PLLP_VALUE](#)(VALUE) (((VALUE) == 2) || ((VALUE) == 4) || ((VALUE) == 6) || ((VALUE) == 8))
- #define [IS_RCC_PLLQ_VALUE](#)(VALUE) ((4 <= (VALUE)) && ((VALUE) <= 15))
- #define [IS_RCC_PLLI2SN_VALUE](#)(VALUE) ((192 <= (VALUE)) && ((VALUE) <= 432))
- #define [IS_RCC_PLLI2SR_VALUE](#)(VALUE) ((2 <= (VALUE)) && ((VALUE) <= 7))

- #define [RCC_SYSCLKSource_HSI](#) ((uint32_t)0x00000000)
- #define [RCC_SYSCLKSource_HSE](#) ((uint32_t)0x00000001)
- #define [RCC_SYSCLKSource_PLLCLK](#) ((uint32_t)0x00000002)
- #define [IS_RCC_SYSCLK_SOURCE](#)(SOURCE)
- #define [RCC_SYSCLK_Div1](#) ((uint32_t)0x00000000)
- #define [RCC_SYSCLK_Div2](#) ((uint32_t)0x00000080)
- #define [RCC_SYSCLK_Div4](#) ((uint32_t)0x00000090)
- #define [RCC_SYSCLK_Div8](#) ((uint32_t)0x000000A0)
- #define [RCC_SYSCLK_Div16](#) ((uint32_t)0x000000B0)
- #define [RCC_SYSCLK_Div64](#) ((uint32_t)0x000000C0)
- #define [RCC_SYSCLK_Div128](#) ((uint32_t)0x000000D0)
- #define [RCC_SYSCLK_Div256](#) ((uint32_t)0x000000E0)
- #define [RCC_SYSCLK_Div512](#) ((uint32_t)0x000000F0)
- #define [IS_RCC_HCLK](#)(HCLK)
- #define [RCC_HCLK_Div1](#) ((uint32_t)0x00000000)
- #define [RCC_HCLK_Div2](#) ((uint32_t)0x00001000)
- #define [RCC_HCLK_Div4](#) ((uint32_t)0x00001400)
- #define [RCC_HCLK_Div8](#) ((uint32_t)0x00001800)
- #define [RCC_HCLK_Div16](#) ((uint32_t)0x00001C00)
- #define [IS_RCC_PCLK](#)(PCLK)
- #define [RCC_IT_LSIRDY](#) ((uint8_t)0x01)
- #define [RCC_IT_LSERDY](#) ((uint8_t)0x02)
- #define [RCC_IT_HSIRDY](#) ((uint8_t)0x04)
- #define [RCC_IT_HSERDY](#) ((uint8_t)0x08)
- #define [RCC_IT_PLLRDY](#) ((uint8_t)0x10)
- #define [RCC_IT_PLLI2SRDY](#) ((uint8_t)0x20)
- #define [RCC_IT_CSS](#) ((uint8_t)0x80)
- #define [IS_RCC_IT](#)(IT) (((IT) & (uint8_t)0xC0) == 0x00) && ((IT) != 0x00)
- #define [IS_RCC_GET_IT](#)(IT)
- #define [IS_RCC_CLEAR_IT](#)(IT) (((IT) & (uint8_t)0x40) == 0x00) && ((IT) != 0x00)
- #define [RCC_LSE_OFF](#) ((uint8_t)0x00)
- #define [RCC_LSE_ON](#) ((uint8_t)0x01)
- #define [RCC_LSE_Bypass](#) ((uint8_t)0x04)
- #define [IS_RCC_LSE](#)(LSE)
- #define [RCC_RTCCLKSource_LSE](#) ((uint32_t)0x00000100)
- #define [RCC_RTCCLKSource_LSI](#) ((uint32_t)0x00000200)
- #define [RCC_RTCCLKSource_HSE_Div2](#) ((uint32_t)0x00020300)
- #define [RCC_RTCCLKSource_HSE_Div3](#) ((uint32_t)0x00030300)
- #define [RCC_RTCCLKSource_HSE_Div4](#) ((uint32_t)0x00040300)
- #define [RCC_RTCCLKSource_HSE_Div5](#) ((uint32_t)0x00050300)
- #define [RCC_RTCCLKSource_HSE_Div6](#) ((uint32_t)0x00060300)
- #define [RCC_RTCCLKSource_HSE_Div7](#) ((uint32_t)0x00070300)
- #define [RCC_RTCCLKSource_HSE_Div8](#) ((uint32_t)0x00080300)
- #define [RCC_RTCCLKSource_HSE_Div9](#) ((uint32_t)0x00090300)
- #define [RCC_RTCCLKSource_HSE_Div10](#) ((uint32_t)0x000A0300)
- #define [RCC_RTCCLKSource_HSE_Div11](#) ((uint32_t)0x000B0300)
- #define [RCC_RTCCLKSource_HSE_Div12](#) ((uint32_t)0x000C0300)
- #define [RCC_RTCCLKSource_HSE_Div13](#) ((uint32_t)0x000D0300)
- #define [RCC_RTCCLKSource_HSE_Div14](#) ((uint32_t)0x000E0300)
- #define [RCC_RTCCLKSource_HSE_Div15](#) ((uint32_t)0x000F0300)
- #define [RCC_RTCCLKSource_HSE_Div16](#) ((uint32_t)0x00100300)
- #define [RCC_RTCCLKSource_HSE_Div17](#) ((uint32_t)0x00110300)
- #define [RCC_RTCCLKSource_HSE_Div18](#) ((uint32_t)0x00120300)
- #define [RCC_RTCCLKSource_HSE_Div19](#) ((uint32_t)0x00130300)
- #define [RCC_RTCCLKSource_HSE_Div20](#) ((uint32_t)0x00140300)

```

• #define RCC_RTCCLKSource_HSE_Div21 ((uint32_t)0x00150300)
• #define RCC_RTCCLKSource_HSE_Div22 ((uint32_t)0x00160300)
• #define RCC_RTCCLKSource_HSE_Div23 ((uint32_t)0x00170300)
• #define RCC_RTCCLKSource_HSE_Div24 ((uint32_t)0x00180300)
• #define RCC_RTCCLKSource_HSE_Div25 ((uint32_t)0x00190300)
• #define RCC_RTCCLKSource_HSE_Div26 ((uint32_t)0x001A0300)
• #define RCC_RTCCLKSource_HSE_Div27 ((uint32_t)0x001B0300)
• #define RCC_RTCCLKSource_HSE_Div28 ((uint32_t)0x001C0300)
• #define RCC_RTCCLKSource_HSE_Div29 ((uint32_t)0x001D0300)
• #define RCC_RTCCLKSource_HSE_Div30 ((uint32_t)0x001E0300)
• #define RCC_RTCCLKSource_HSE_Div31 ((uint32_t)0x001F0300)
• #define IS_RCC_RTCCLK_SOURCE(SOURCE)
• #define RCC_I2S2CLKSource_PLLI2S ((uint8_t)0x00)
• #define RCC_I2S2CLKSource_Ext ((uint8_t)0x01)
• #define IS_RCC_I2SCLK_SOURCE(SOURCE) (((SOURCE) == RCC_I2S2CLKSource_PLLI2S) ||
((SOURCE) == RCC_I2S2CLKSource_Ext))
• #define RCC_AHB1Periph_GPIOA ((uint32_t)0x00000001)
• #define RCC_AHB1Periph_GPIOB ((uint32_t)0x00000002)
• #define RCC_AHB1Periph_GPIOC ((uint32_t)0x00000004)
• #define RCC_AHB1Periph_GPIOD ((uint32_t)0x00000008)
• #define RCC_AHB1Periph_GPIOE ((uint32_t)0x00000010)
• #define RCC_AHB1Periph_GPIOF ((uint32_t)0x00000020)
• #define RCC_AHB1Periph_GPIOG ((uint32_t)0x00000040)
• #define RCC_AHB1Periph_GPIOH ((uint32_t)0x00000080)
• #define RCC_AHB1Periph_GPIOI ((uint32_t)0x00000100)
• #define RCC_AHB1Periph_CRC ((uint32_t)0x00001000)
• #define RCC_AHB1Periph_FLITF ((uint32_t)0x00008000)
• #define RCC_AHB1Periph_SRAM1 ((uint32_t)0x00010000)
• #define RCC_AHB1Periph_SRAM2 ((uint32_t)0x00020000)
• #define RCC_AHB1Periph_BKPSRAM ((uint32_t)0x00040000)
• #define RCC_AHB1Periph_CCMDATARAMEN ((uint32_t)0x00100000)
• #define RCC_AHB1Periph_DMA1 ((uint32_t)0x00200000)
• #define RCC_AHB1Periph_DMA2 ((uint32_t)0x00400000)
• #define RCC_AHB1Periph_ETH_MAC ((uint32_t)0x02000000)
• #define RCC_AHB1Periph_ETH_MAC_Tx ((uint32_t)0x04000000)
• #define RCC_AHB1Periph_ETH_MAC_Rx ((uint32_t)0x08000000)
• #define RCC_AHB1Periph_ETH_MAC_PTP ((uint32_t)0x10000000)
• #define RCC_AHB1Periph_OTG_HS ((uint32_t)0x20000000)
• #define RCC_AHB1Periph_OTG_HS_ULPI ((uint32_t)0x40000000)
• #define IS_RCC_AHB1_CLOCK_PERIPH(PERIPH) (((PERIPH) & 0x818BEE00) == 0x00) && ((PERIPH) !=
0x00))
• #define IS_RCC_AHB1_RESET_PERIPH(PERIPH) (((PERIPH) & 0xDD9FEE00) == 0x00) && ((PERIPH)
!= 0x00))
• #define IS_RCC_AHB1_LPMODE_PERIPH(PERIPH) (((PERIPH) & 0x81986E00) == 0x00) && ((PERIPH)
!= 0x00))
• #define RCC_AHB2Periph_DCM1 ((uint32_t)0x00000001)
• #define RCC_AHB2Periph_Cryp ((uint32_t)0x00000010)
• #define RCC_AHB2Periph_HASH ((uint32_t)0x00000020)
• #define RCC_AHB2Periph_RNG ((uint32_t)0x00000040)
• #define RCC_AHB2Periph_OTG_FS ((uint32_t)0x00000080)
• #define IS_RCC_AHB2_PERIPH(PERIPH) (((PERIPH) & 0xFFFFF0E) == 0x00) && ((PERIPH) != 0x00))
• #define RCC_AHB3Periph_FSMC ((uint32_t)0x00000001)
• #define IS_RCC_AHB3_PERIPH(PERIPH) (((PERIPH) & 0xFFFFF0E) == 0x00) && ((PERIPH) != 0x00))
• #define RCC_APB1Periph_TIM2 ((uint32_t)0x00000001)
• #define RCC_APB1Periph_TIM3 ((uint32_t)0x00000002)

```

- #define `RCC_APB1Periph_TIM4` ((uint32_t)0x00000004)
- #define `RCC_APB1Periph_TIM5` ((uint32_t)0x00000008)
- #define `RCC_APB1Periph_TIM6` ((uint32_t)0x00000010)
- #define `RCC_APB1Periph_TIM7` ((uint32_t)0x00000020)
- #define `RCC_APB1Periph_TIM12` ((uint32_t)0x00000040)
- #define `RCC_APB1Periph_TIM13` ((uint32_t)0x00000080)
- #define `RCC_APB1Periph_TIM14` ((uint32_t)0x00000100)
- #define `RCC_APB1Periph_WWDG` ((uint32_t)0x00000800)
- #define `RCC_APB1Periph_SPI2` ((uint32_t)0x00004000)
- #define `RCC_APB1Periph_SPI3` ((uint32_t)0x00008000)
- #define `RCC_APB1Periph_USART2` ((uint32_t)0x00020000)
- #define `RCC_APB1Periph_USART3` ((uint32_t)0x00040000)
- #define `RCC_APB1Periph_UART4` ((uint32_t)0x00080000)
- #define `RCC_APB1Periph_UART5` ((uint32_t)0x00100000)
- #define `RCC_APB1Periph_I2C1` ((uint32_t)0x00200000)
- #define `RCC_APB1Periph_I2C2` ((uint32_t)0x00400000)
- #define `RCC_APB1Periph_I2C3` ((uint32_t)0x00800000)
- #define `RCC_APB1Periph_CAN1` ((uint32_t)0x02000000)
- #define `RCC_APB1Periph_CAN2` ((uint32_t)0x04000000)
- #define `RCC_APB1Periph_PWR` ((uint32_t)0x10000000)
- #define `RCC_APB1Periph_DAC` ((uint32_t)0x20000000)
- #define `IS_RCC_APB1_PERIPH(PERIPH)` (((PERIPH) & 0xC9013600) == 0x00) && ((PERIPH) != 0x00)
- #define `RCC_APB2Periph_TIM1` ((uint32_t)0x00000001)
- #define `RCC_APB2Periph_TIM8` ((uint32_t)0x00000002)
- #define `RCC_APB2Periph_USART1` ((uint32_t)0x00000010)
- #define `RCC_APB2Periph_USART6` ((uint32_t)0x00000020)
- #define `RCC_APB2Periph_ADC` ((uint32_t)0x00000100)
- #define `RCC_APB2Periph_ADC1` ((uint32_t)0x00000100)
- #define `RCC_APB2Periph_ADC2` ((uint32_t)0x00000200)
- #define `RCC_APB2Periph_ADC3` ((uint32_t)0x00000400)
- #define `RCC_APB2Periph_SDIO` ((uint32_t)0x00000800)
- #define `RCC_APB2Periph_SPI1` ((uint32_t)0x00001000)
- #define `RCC_APB2Periph_SYSCFG` ((uint32_t)0x00004000)
- #define `RCC_APB2Periph_TIM9` ((uint32_t)0x00010000)
- #define `RCC_APB2Periph_TIM10` ((uint32_t)0x00020000)
- #define `RCC_APB2Periph_TIM11` ((uint32_t)0x00040000)
- #define `IS_RCC_APB2_PERIPH(PERIPH)` (((PERIPH) & 0xFFF8A0CC) == 0x00) && ((PERIPH) != 0x00)
- #define `IS_RCC_APB2_RESET_PERIPH(PERIPH)` (((PERIPH) & 0xFFF8A6CC) == 0x00) && ((PERIPH) != 0x00)
- #define `RCC_MCO1Source_HSI` ((uint32_t)0x00000000)
- #define `RCC_MCO1Source_LSE` ((uint32_t)0x00200000)
- #define `RCC_MCO1Source_HSE` ((uint32_t)0x00400000)
- #define `RCC_MCO1Source_PLLCLK` ((uint32_t)0x00600000)
- #define `RCC_MCO1Div_1` ((uint32_t)0x00000000)
- #define `RCC_MCO1Div_2` ((uint32_t)0x04000000)
- #define `RCC_MCO1Div_3` ((uint32_t)0x05000000)
- #define `RCC_MCO1Div_4` ((uint32_t)0x06000000)
- #define `RCC_MCO1Div_5` ((uint32_t)0x07000000)
- #define `IS_RCC_MCO1SOURCE(SOURCE)`
- #define `IS_RCC_MCO1DIV(DIV)`
- #define `RCC_MCO2Source_SYSCLK` ((uint32_t)0x00000000)
- #define `RCC_MCO2Source_PLLI2SCLK` ((uint32_t)0x40000000)
- #define `RCC_MCO2Source_HSE` ((uint32_t)0x80000000)
- #define `RCC_MCO2Source_PLLCLK` ((uint32_t)0xC0000000)
- #define `RCC_MCO2Div_1` ((uint32_t)0x00000000)

- `#define RCC_MCO2Div_2 ((uint32_t)0x20000000)`
- `#define RCC_MCO2Div_3 ((uint32_t)0x28000000)`
- `#define RCC_MCO2Div_4 ((uint32_t)0x30000000)`
- `#define RCC_MCO2Div_5 ((uint32_t)0x38000000)`
- `#define IS_RCC_MCO2SOURCE(SOURCE)`
- `#define IS_RCC_MCO2DIV(DIV)`
- `#define RCC_FLAG_HSIRDY ((uint8_t)0x21)`
- `#define RCC_FLAG_HSERDY ((uint8_t)0x31)`
- `#define RCC_FLAG_PLLRDY ((uint8_t)0x39)`
- `#define RCC_FLAG_PLLI2SRDY ((uint8_t)0x3B)`
- `#define RCC_FLAG_LSERDY ((uint8_t)0x41)`
- `#define RCC_FLAG_LSIRDY ((uint8_t)0x61)`
- `#define RCC_FLAG BORRST ((uint8_t)0x79)`
- `#define RCC_FLAG_PINRST ((uint8_t)0x7A)`
- `#define RCC_FLAG_PORRST ((uint8_t)0x7B)`
- `#define RCC_FLAG_SFTRST ((uint8_t)0x7C)`
- `#define RCC_FLAG_IWDGRST ((uint8_t)0x7D)`
- `#define RCC_FLAG_WWDGRST ((uint8_t)0x7E)`
- `#define RCC_FLAG_LPWRST ((uint8_t)0x7F)`
- `#define IS_RCC_FLAG(FLAG)`
- `#define IS_RCC_CALIBRATION_VALUE(VALUE) ((VALUE) <= 0x1F)`

Functions

- void `RCC_DeInit` (void)
Resets the RCC clock configuration to the default reset state.
- void `RCC_HSEConfig` (uint8_t RCC_HSE)
Configures the External High Speed oscillator (HSE).
- ErrorStatus `RCC_WaitForHSEStartUp` (void)
Waits for HSE start-up.
- void `RCC_AdjustHSICalibrationValue` (uint8_t HSICalibrationValue)
Adjusts the Internal High Speed oscillator (HSI) calibration value.
- void `RCC_HSICmd` (FunctionalState NewState)
Enables or disables the Internal High Speed oscillator (HSI).
- void `RCC_LSEConfig` (uint8_t RCC_LSE)
Configures the External Low Speed oscillator (LSE).
- void `RCC_LSICmd` (FunctionalState NewState)
Enables or disables the Internal Low Speed oscillator (LSI).
- void `RCC_PLLConfig` (uint32_t RCC_PLLSource, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP, uint32_t PLLQ)
Configures the main PLL clock source, multiplication and division factors.
- void `RCC_PLLCmd` (FunctionalState NewState)
Enables or disables the main PLL.
- void `RCC_PLLI2SConfig` (uint32_t PLLI2SN, uint32_t PLLI2SR)
Configures the PLLI2S clock multiplication and division factors.
- void `RCC_PLLI2SCmd` (FunctionalState NewState)
Enables or disables the PLLI2S.
- void `RCC_ClockSecuritySystemCmd` (FunctionalState NewState)
Enables or disables the Clock Security System.
- void `RCC_MCO1Config` (uint32_t RCC_MCO1Source, uint32_t RCC_MCO1Div)
Selects the clock source to output on MCO1 pin(PA8).
- void `RCC_MCO2Config` (uint32_t RCC_MCO2Source, uint32_t RCC_MCO2Div)
Selects the clock source to output on MCO2 pin(PC9).

- void [RCC_SYSCLKConfig](#) (uint32_t RCC_SYSCLKSource)
Configures the system clock (SYSCLK).
- uint8_t [RCC_GetSYSCLKSource](#) (void)
Returns the clock source used as system clock.
- void [RCC_HCLKConfig](#) (uint32_t RCC_SYSCLK)
Configures the AHB clock (HCLK).
- void [RCC_PCLK1Config](#) (uint32_t RCC_HCLK)
Configures the Low Speed APB clock (PCLK1).
- void [RCC_PCLK2Config](#) (uint32_t RCC_HCLK)
Configures the High Speed APB clock (PCLK2).
- void [RCC_GetClocksFreq](#) (RCC_ClocksTypeDef *RCC_Clocks)
Returns the frequencies of different on chip clocks; SYSCLK, HCLK, PCLK1 and PCLK2.

- void [RCC_RTCCLKConfig](#) (uint32_t RCC_RTCCLKSource)
Configures the RTC clock (RTCCLK).
- void [RCC_RTCCLKCmd](#) (FunctionalState NewState)
Enables or disables the RTC clock.
- void [RCC_BackupResetCmd](#) (FunctionalState NewState)
Forces or releases the Backup domain reset.
- void [RCC_I2SCLKConfig](#) (uint32_t RCC_I2SCLKSource)
Configures the I2S clock source (I2SCLK).
- void [RCC_AHB1PeriphClockCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
Enables or disables the AHB1 peripheral clock.
- void [RCC_AHB2PeriphClockCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
Enables or disables the AHB2 peripheral clock.
- void [RCC_AHB3PeriphClockCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Enables or disables the AHB3 peripheral clock.
- void [RCC_APB1PeriphClockCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Enables or disables the Low Speed APB (APB1) peripheral clock.
- void [RCC_APB2PeriphClockCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Enables or disables the High Speed APB (APB2) peripheral clock.
- void [RCC_AHB1PeriphResetCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
Forces or releases AHB1 peripheral reset.
- void [RCC_AHB2PeriphResetCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
Forces or releases AHB2 peripheral reset.
- void [RCC_AHB3PeriphResetCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Forces or releases AHB3 peripheral reset.
- void [RCC_APB1PeriphResetCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Forces or releases Low Speed APB (APB1) peripheral reset.
- void [RCC_APB2PeriphResetCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Forces or releases High Speed APB (APB2) peripheral reset.
- void [RCC_AHB1PeriphClockLPModeCmd](#) (uint32_t RCC_AHB1Periph, FunctionalState NewState)
Enables or disables the AHB1 peripheral clock during Low Power (Sleep) mode.
- void [RCC_AHB2PeriphClockLPModeCmd](#) (uint32_t RCC_AHB2Periph, FunctionalState NewState)
Enables or disables the AHB2 peripheral clock during Low Power (Sleep) mode.
- void [RCC_AHB3PeriphClockLPModeCmd](#) (uint32_t RCC_AHB3Periph, FunctionalState NewState)
Enables or disables the AHB3 peripheral clock during Low Power (Sleep) mode.
- void [RCC_APB1PeriphClockLPModeCmd](#) (uint32_t RCC_APB1Periph, FunctionalState NewState)
Enables or disables the APB1 peripheral clock during Low Power (Sleep) mode.
- void [RCC_APB2PeriphClockLPModeCmd](#) (uint32_t RCC_APB2Periph, FunctionalState NewState)
Enables or disables the APB2 peripheral clock during Low Power (Sleep) mode.

- void [RCC_ITConfig](#) (uint8_t RCC_IT, FunctionalState NewState)
Enables or disables the specified RCC interrupts.
- FlagStatus [RCC_GetFlagStatus](#) (uint8_t RCC_FLAG)
Checks whether the specified RCC flag is set or not.
- void [RCC_ClearFlag](#) (void)
Clears the RCC reset flags. The reset flags are: RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST, RCC_FLAG_LPWRST.
- ITStatus [RCC_GetITStatus](#) (uint8_t RCC_IT)
Checks whether the specified RCC interrupt has occurred or not.
- void [RCC_ClearITPendingBit](#) (uint8_t RCC_IT)
Clears the RCC's interrupt pending bits.

6.25.1 Detailed Description

This file contains all the functions prototypes for the RCC firmware library.

Author

MCD Application Team

Version

V1.0.0

Date

30-September-2011

Attention

THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

© COPYRIGHT 2011 STMicroelectronics

6.26 stm32f4xx_rcc.h

[Go to the documentation of this file.](#)

```

00001
00022 /* Define to prevent recursive inclusion -----*/
00023 #ifndef __STM32F4xx_RCC_H
00024 #define __STM32F4xx_RCC_H
00025
00026 #ifdef __cplusplus
00027     extern "C" {
00028 #endif
00029
00030 /* Includes -----*/
00031 #include "stm32f4xx.h"
00032
00041 /* Exported types -----*/
00042 typedef struct
00043 {
00044     uint32_t  SYSCLK_Frequency;
00045     uint32_t  HCLK_Frequency;
00046     uint32_t  PCLK1_Frequency;
00047     uint32_t  PCLK2_Frequency;

```



```

00048 }RCC_ClocksTypeDef;
00049
00050 /* Exported constants -----*/
00051
00059 #define RCC_HSE_OFF ((uint8_t)0x00)
00060 #define RCC_HSE_ON ((uint8_t)0x01)
00061 #define RCC_HSE_Bypass ((uint8_t)0x05)
00062 #define IS_RCC_HSE(HSE) (((HSE) == RCC_HSE_OFF) || ((HSE) == RCC_HSE_ON) || \
00063 ((HSE) == RCC_HSE_Bypass))
00071 #define RCC_PLLSource_HSI ((uint32_t)0x00000000)
00072 #define RCC_PLLSource_HSE ((uint32_t)0x00400000)
00073 #define IS_RCC_PLL_SOURCE(SOURCE) (((SOURCE) == RCC_PLLSource_HSI) || \
00074 ((SOURCE) == RCC_PLLSource_HSE))
00075 #define IS_RCC_PLLM_VALUE(VALUE) ((VALUE) <= 63)
00076 #define IS_RCC_PLLN_VALUE(VALUE) ((192 <= (VALUE)) && ((VALUE) <= 432))
00077 #define IS_RCC_PLLP_VALUE(VALUE) (((VALUE) == 2) || ((VALUE) == 4) || ((VALUE) == 6) || ((VALUE) ==
8))
00078 #define IS_RCC_PLLQ_VALUE(VALUE) ((4 <= (VALUE)) && ((VALUE) <= 15))
00079
00080 #define IS_RCC_PLLI2SN_VALUE(VALUE) ((192 <= (VALUE)) && ((VALUE) <= 432))
00081 #define IS_RCC_PLLI2SR_VALUE(VALUE) ((2 <= (VALUE)) && ((VALUE) <= 7))
00089 #define RCC_SYSCLKSource_HSI ((uint32_t)0x00000000)
00090 #define RCC_SYSCLKSource_HSE ((uint32_t)0x00000001)
00091 #define RCC_SYSCLKSource_PLLCLK ((uint32_t)0x00000002)
00092 #define IS_RCC_SYSCLK_SOURCE(SOURCE) (((SOURCE) == RCC_SYSCLKSource_HSI) || \
00093 ((SOURCE) == RCC_SYSCLKSource_HSE) || \
00094 ((SOURCE) == RCC_SYSCLKSource_PLLCLK))
00102 #define RCC_SYSCLK_Div1 ((uint32_t)0x00000000)
00103 #define RCC_SYSCLK_Div2 ((uint32_t)0x00000080)
00104 #define RCC_SYSCLK_Div4 ((uint32_t)0x00000090)
00105 #define RCC_SYSCLK_Div8 ((uint32_t)0x000000A0)
00106 #define RCC_SYSCLK_Div16 ((uint32_t)0x000000B0)
00107 #define RCC_SYSCLK_Div64 ((uint32_t)0x000000C0)
00108 #define RCC_SYSCLK_Div128 ((uint32_t)0x000000D0)
00109 #define RCC_SYSCLK_Div256 ((uint32_t)0x000000E0)
00110 #define RCC_SYSCLK_Div512 ((uint32_t)0x000000F0)
00111 #define IS_RCC_HCLK(HCLK) (((HCLK) == RCC_SYSCLK_Div1) || ((HCLK) == RCC_SYSCLK_Div2) || \
00112 ((HCLK) == RCC_SYSCLK_Div4) || ((HCLK) == RCC_SYSCLK_Div8) || \
00113 ((HCLK) == RCC_SYSCLK_Div16) || ((HCLK) == RCC_SYSCLK_Div64) || \
00114 ((HCLK) == RCC_SYSCLK_Div128) || ((HCLK) == RCC_SYSCLK_Div256) || \
00115 ((HCLK) == RCC_SYSCLK_Div512))
00123 #define RCC_HCLK_Div1 ((uint32_t)0x00000000)
00124 #define RCC_HCLK_Div2 ((uint32_t)0x00001000)
00125 #define RCC_HCLK_Div4 ((uint32_t)0x00001400)
00126 #define RCC_HCLK_Div8 ((uint32_t)0x00001800)
00127 #define RCC_HCLK_Div16 ((uint32_t)0x00001C00)
00128 #define IS_RCC_PCLK(PCLK) (((PCLK) == RCC_HCLK_Div1) || ((PCLK) == RCC_HCLK_Div2) || \
00129 ((PCLK) == RCC_HCLK_Div4) || ((PCLK) == RCC_HCLK_Div8) || \
00130 ((PCLK) == RCC_HCLK_Div16))
00138 #define RCC_IT_LSIRDY ((uint8_t)0x01)
00139 #define RCC_IT_LSERDY ((uint8_t)0x02)
00140 #define RCC_IT_HSIRDY ((uint8_t)0x04)
00141 #define RCC_IT_HSERDY ((uint8_t)0x08)
00142 #define RCC_IT_PLLRDY ((uint8_t)0x10)
00143 #define RCC_IT_PLLI2SRDY ((uint8_t)0x20)
00144 #define RCC_IT_CSS ((uint8_t)0x80)
00145 #define IS_RCC_IT(IT) (((IT) & (uint8_t)0xC0) == 0x00) && ((IT) != 0x00)
00146 #define IS_RCC_GET_IT(IT) (((IT) == RCC_IT_LSIRDY) || ((IT) == RCC_IT_LSERDY) || \
00147 ((IT) == RCC_IT_HSIRDY) || ((IT) == RCC_IT_HSERDY) || \
00148 ((IT) == RCC_IT_PLLRDY) || ((IT) == RCC_IT_CSS) || \
00149 ((IT) == RCC_IT_PLLI2SRDY))
00150 #define IS_RCC_CLEAR_IT(IT) (((IT) & (uint8_t)0x40) == 0x00) && ((IT) != 0x00)
00158 #define RCC_LSE_OFF ((uint8_t)0x00)
00159 #define RCC_LSE_ON ((uint8_t)0x01)
00160 #define RCC_LSE_Bypass ((uint8_t)0x04)
00161 #define IS_RCC_LSE(LSE) (((LSE) == RCC_LSE_OFF) || ((LSE) == RCC_LSE_ON) || \
00162 ((LSE) == RCC_LSE_Bypass))
00170 #define RCC_RTCCLKSource_LSE ((uint32_t)0x00000100)
00171 #define RCC_RTCCLKSource_LSI ((uint32_t)0x00000200)
00172 #define RCC_RTCCLKSource_HSE_Div2 ((uint32_t)0x00020300)
00173 #define RCC_RTCCLKSource_HSE_Div3 ((uint32_t)0x00030300)
00174 #define RCC_RTCCLKSource_HSE_Div4 ((uint32_t)0x00040300)
00175 #define RCC_RTCCLKSource_HSE_Div5 ((uint32_t)0x00050300)
00176 #define RCC_RTCCLKSource_HSE_Div6 ((uint32_t)0x00060300)
00177 #define RCC_RTCCLKSource_HSE_Div7 ((uint32_t)0x00070300)
00178 #define RCC_RTCCLKSource_HSE_Div8 ((uint32_t)0x00080300)
00179 #define RCC_RTCCLKSource_HSE_Div9 ((uint32_t)0x00090300)
00180 #define RCC_RTCCLKSource_HSE_Div10 ((uint32_t)0x000A0300)
00181 #define RCC_RTCCLKSource_HSE_Div11 ((uint32_t)0x000B0300)
00182 #define RCC_RTCCLKSource_HSE_Div12 ((uint32_t)0x000C0300)
00183 #define RCC_RTCCLKSource_HSE_Div13 ((uint32_t)0x000D0300)
00184 #define RCC_RTCCLKSource_HSE_Div14 ((uint32_t)0x000E0300)
00185 #define RCC_RTCCLKSource_HSE_Div15 ((uint32_t)0x000F0300)
00186 #define RCC_RTCCLKSource_HSE_Div16 ((uint32_t)0x00100300)
00187 #define RCC_RTCCLKSource_HSE_Div17 ((uint32_t)0x00110300)
00188 #define RCC_RTCCLKSource_HSE_Div18 ((uint32_t)0x00120300)
00189 #define RCC_RTCCLKSource_HSE_Div19 ((uint32_t)0x00130300)

```

```

00190 #define RCC_RTCCLKSource_HSE_Div20 ((uint32_t)0x00140300)
00191 #define RCC_RTCCLKSource_HSE_Div21 ((uint32_t)0x00150300)
00192 #define RCC_RTCCLKSource_HSE_Div22 ((uint32_t)0x00160300)
00193 #define RCC_RTCCLKSource_HSE_Div23 ((uint32_t)0x00170300)
00194 #define RCC_RTCCLKSource_HSE_Div24 ((uint32_t)0x00180300)
00195 #define RCC_RTCCLKSource_HSE_Div25 ((uint32_t)0x00190300)
00196 #define RCC_RTCCLKSource_HSE_Div26 ((uint32_t)0x001A0300)
00197 #define RCC_RTCCLKSource_HSE_Div27 ((uint32_t)0x001B0300)
00198 #define RCC_RTCCLKSource_HSE_Div28 ((uint32_t)0x001C0300)
00199 #define RCC_RTCCLKSource_HSE_Div29 ((uint32_t)0x001D0300)
00200 #define RCC_RTCCLKSource_HSE_Div30 ((uint32_t)0x001E0300)
00201 #define RCC_RTCCLKSource_HSE_Div31 ((uint32_t)0x001F0300)
00202 #define IS_RCC_RTCCLK_SOURCE(SOURCE) (((SOURCE) == RCC_RTCCLKSource_LSE) || \
00203 ((SOURCE) == RCC_RTCCLKSource_LSI) || \
00204 ((SOURCE) == RCC_RTCCLKSource_HSE_Div2) || \
00205 ((SOURCE) == RCC_RTCCLKSource_HSE_Div3) || \
00206 ((SOURCE) == RCC_RTCCLKSource_HSE_Div4) || \
00207 ((SOURCE) == RCC_RTCCLKSource_HSE_Div5) || \
00208 ((SOURCE) == RCC_RTCCLKSource_HSE_Div6) || \
00209 ((SOURCE) == RCC_RTCCLKSource_HSE_Div7) || \
00210 ((SOURCE) == RCC_RTCCLKSource_HSE_Div8) || \
00211 ((SOURCE) == RCC_RTCCLKSource_HSE_Div9) || \
00212 ((SOURCE) == RCC_RTCCLKSource_HSE_Div10) || \
00213 ((SOURCE) == RCC_RTCCLKSource_HSE_Div11) || \
00214 ((SOURCE) == RCC_RTCCLKSource_HSE_Div12) || \
00215 ((SOURCE) == RCC_RTCCLKSource_HSE_Div13) || \
00216 ((SOURCE) == RCC_RTCCLKSource_HSE_Div14) || \
00217 ((SOURCE) == RCC_RTCCLKSource_HSE_Div15) || \
00218 ((SOURCE) == RCC_RTCCLKSource_HSE_Div16) || \
00219 ((SOURCE) == RCC_RTCCLKSource_HSE_Div17) || \
00220 ((SOURCE) == RCC_RTCCLKSource_HSE_Div18) || \
00221 ((SOURCE) == RCC_RTCCLKSource_HSE_Div19) || \
00222 ((SOURCE) == RCC_RTCCLKSource_HSE_Div20) || \
00223 ((SOURCE) == RCC_RTCCLKSource_HSE_Div21) || \
00224 ((SOURCE) == RCC_RTCCLKSource_HSE_Div22) || \
00225 ((SOURCE) == RCC_RTCCLKSource_HSE_Div23) || \
00226 ((SOURCE) == RCC_RTCCLKSource_HSE_Div24) || \
00227 ((SOURCE) == RCC_RTCCLKSource_HSE_Div25) || \
00228 ((SOURCE) == RCC_RTCCLKSource_HSE_Div26) || \
00229 ((SOURCE) == RCC_RTCCLKSource_HSE_Div27) || \
00230 ((SOURCE) == RCC_RTCCLKSource_HSE_Div28) || \
00231 ((SOURCE) == RCC_RTCCLKSource_HSE_Div29) || \
00232 ((SOURCE) == RCC_RTCCLKSource_HSE_Div30) || \
00233 ((SOURCE) == RCC_RTCCLKSource_HSE_Div31))
00241 #define RCC_I2S2CLKSource_PLLI2S ((uint8_t)0x00)
00242 #define RCC_I2S2CLKSource_Ext ((uint8_t)0x01)
00243
00244 #define IS_RCC_I2SCLK_SOURCE(SOURCE) (((SOURCE) == RCC_I2S2CLKSource_PLLI2S) || ((SOURCE) ==
RCC_I2S2CLKSource_Ext))
00252 #define RCC_AHB1Periph_GPIOA ((uint32_t)0x00000001)
00253 #define RCC_AHB1Periph_GPIOB ((uint32_t)0x00000002)
00254 #define RCC_AHB1Periph_GPIOC ((uint32_t)0x00000004)
00255 #define RCC_AHB1Periph_GPIOD ((uint32_t)0x00000008)
00256 #define RCC_AHB1Periph_GPIOE ((uint32_t)0x00000010)
00257 #define RCC_AHB1Periph_GPIOF ((uint32_t)0x00000020)
00258 #define RCC_AHB1Periph_GPIOG ((uint32_t)0x00000040)
00259 #define RCC_AHB1Periph_GPIOH ((uint32_t)0x00000080)
00260 #define RCC_AHB1Periph_GPIOI ((uint32_t)0x00000100)
00261 #define RCC_AHB1Periph_CRC ((uint32_t)0x00001000)
00262 #define RCC_AHB1Periph_FLITF ((uint32_t)0x00008000)
00263 #define RCC_AHB1Periph_SRAM1 ((uint32_t)0x00010000)
00264 #define RCC_AHB1Periph_SRAM2 ((uint32_t)0x00020000)
00265 #define RCC_AHB1Periph_BKPSRAM ((uint32_t)0x00040000)
00266 #define RCC_AHB1Periph_CCMDATARAMEN ((uint32_t)0x00100000)
00267 #define RCC_AHB1Periph_DMA1 ((uint32_t)0x00200000)
00268 #define RCC_AHB1Periph_DMA2 ((uint32_t)0x00400000)
00269 #define RCC_AHB1Periph_ETH_MAC ((uint32_t)0x02000000)
00270 #define RCC_AHB1Periph_ETH_MAC_Tx ((uint32_t)0x04000000)
00271 #define RCC_AHB1Periph_ETH_MAC_Rx ((uint32_t)0x08000000)
00272 #define RCC_AHB1Periph_ETH_MAC_PTP ((uint32_t)0x10000000)
00273 #define RCC_AHB1Periph_OTG_HS ((uint32_t)0x20000000)
00274 #define RCC_AHB1Periph_OTG_HS_ULPI ((uint32_t)0x40000000)
00275 #define IS_RCC_AHB1_CLOCK_PERIPH(PERIPH) (((PERIPH) & 0x818BEE00) == 0x00) && ((PERIPH) != 0x00)
00276 #define IS_RCC_AHB1_RESET_PERIPH(PERIPH) (((PERIPH) & 0xDD9FEE00) == 0x00) && ((PERIPH) != 0x00)
00277 #define IS_RCC_AHB1_LPMODE_PERIPH(PERIPH) (((PERIPH) & 0x81986E00) == 0x00) && ((PERIPH) != 0x00)
00285 #define RCC_AHB2Periph_DCM1 ((uint32_t)0x00000001)
00286 #define RCC_AHB2Periph_CRYP ((uint32_t)0x00000010)
00287 #define RCC_AHB2Periph_HASH ((uint32_t)0x00000020)
00288 #define RCC_AHB2Periph_RNG ((uint32_t)0x00000040)
00289 #define RCC_AHB2Periph_OTG_FS ((uint32_t)0x00000080)
00290 #define IS_RCC_AHB2_PERIPH(PERIPH) (((PERIPH) & 0xFFFFF0E) == 0x00) && ((PERIPH) != 0x00)
00298 #define RCC_AHB3Periph_FSMC ((uint32_t)0x00000001)
00299 #define IS_RCC_AHB3_PERIPH(PERIPH) (((PERIPH) & 0xFFFFF0FE) == 0x00) && ((PERIPH) != 0x00)
00307 #define RCC_APB1Periph_TIM2 ((uint32_t)0x00000001)
00308 #define RCC_APB1Periph_TIM3 ((uint32_t)0x00000002)
00309 #define RCC_APB1Periph_TIM4 ((uint32_t)0x00000004)
00310 #define RCC_APB1Periph_TIM5 ((uint32_t)0x00000008)

```

```

00311 #define RCC_APB1Periph_TIM6                ((uint32_t)0x00000010)
00312 #define RCC_APB1Periph_TIM7                ((uint32_t)0x00000020)
00313 #define RCC_APB1Periph_TIM12               ((uint32_t)0x00000040)
00314 #define RCC_APB1Periph_TIM13               ((uint32_t)0x00000080)
00315 #define RCC_APB1Periph_TIM14               ((uint32_t)0x00000100)
00316 #define RCC_APB1Periph_WWDG               ((uint32_t)0x00000800)
00317 #define RCC_APB1Periph_SPI2                ((uint32_t)0x00004000)
00318 #define RCC_APB1Periph_SPI3                ((uint32_t)0x00008000)
00319 #define RCC_APB1Periph_USART2              ((uint32_t)0x00020000)
00320 #define RCC_APB1Periph_USART3              ((uint32_t)0x00040000)
00321 #define RCC_APB1Periph_UART4              ((uint32_t)0x00080000)
00322 #define RCC_APB1Periph_UART5              ((uint32_t)0x00100000)
00323 #define RCC_APB1Periph_I2C1                ((uint32_t)0x00200000)
00324 #define RCC_APB1Periph_I2C2                ((uint32_t)0x00400000)
00325 #define RCC_APB1Periph_I2C3                ((uint32_t)0x00800000)
00326 #define RCC_APB1Periph_CAN1                ((uint32_t)0x02000000)
00327 #define RCC_APB1Periph_CAN2                ((uint32_t)0x04000000)
00328 #define RCC_APB1Periph_PWR                ((uint32_t)0x10000000)
00329 #define RCC_APB1Periph_DAC                ((uint32_t)0x20000000)
00330 #define IS_RCC_APB1_PERIPH(PERIPH) (((PERIPH) & 0xc9013600) == 0x00) && ((PERIPH) != 0x00)
00338 #define RCC_APB2Periph_TIM1                ((uint32_t)0x00000001)
00339 #define RCC_APB2Periph_TIM8                ((uint32_t)0x00000002)
00340 #define RCC_APB2Periph_USART1              ((uint32_t)0x00000010)
00341 #define RCC_APB2Periph_USART6              ((uint32_t)0x00000020)
00342 #define RCC_APB2Periph_ADC                ((uint32_t)0x00000100)
00343 #define RCC_APB2Periph_ADC1                ((uint32_t)0x00000100)
00344 #define RCC_APB2Periph_ADC2                ((uint32_t)0x00000200)
00345 #define RCC_APB2Periph_ADC3                ((uint32_t)0x00000400)
00346 #define RCC_APB2Periph_SDIO                ((uint32_t)0x00000800)
00347 #define RCC_APB2Periph_SPI1                ((uint32_t)0x00001000)
00348 #define RCC_APB2Periph_SYSCFG              ((uint32_t)0x00004000)
00349 #define RCC_APB2Periph_TIM9                ((uint32_t)0x00010000)
00350 #define RCC_APB2Periph_TIM10               ((uint32_t)0x00020000)
00351 #define RCC_APB2Periph_TIM11               ((uint32_t)0x00040000)
00352 #define IS_RCC_APB2_PERIPH(PERIPH) (((PERIPH) & 0xffff8a0cc) == 0x00) && ((PERIPH) != 0x00)
00353 #define IS_RCC_APB2_RESET_PERIPH(PERIPH) (((PERIPH) & 0xffff8a6cc) == 0x00) && ((PERIPH) != 0x00)
00361 #define RCC_MC01Source_HSI                 ((uint32_t)0x00000000)
00362 #define RCC_MC01Source_LSE                 ((uint32_t)0x00200000)
00363 #define RCC_MC01Source_HSE                 ((uint32_t)0x00400000)
00364 #define RCC_MC01Source_PLLCLK              ((uint32_t)0x00600000)
00365 #define RCC_MC01Div_1                      ((uint32_t)0x00000000)
00366 #define RCC_MC01Div_2                      ((uint32_t)0x04000000)
00367 #define RCC_MC01Div_3                      ((uint32_t)0x05000000)
00368 #define RCC_MC01Div_4                      ((uint32_t)0x06000000)
00369 #define RCC_MC01Div_5                      ((uint32_t)0x07000000)
00370 #define IS_RCC_MC01SOURCE(SOURCE) (((SOURCE) == RCC_MC01Source_HSI) || ((SOURCE) ==
RCC_MC01Source_LSE) || \
00371                                     ((SOURCE) == RCC_MC01Source_HSE) || ((SOURCE) ==
RCC_MC01Source_PLLCLK))
00372
00373 #define IS_RCC_MC01DIV(DIV) (((DIV) == RCC_MC01Div_1) || ((DIV) == RCC_MC01Div_2) || \
00374                             ((DIV) == RCC_MC01Div_3) || ((DIV) == RCC_MC01Div_4) || \
00375                             ((DIV) == RCC_MC01Div_5))
00383 #define RCC_MC02Source_SYSCLK              ((uint32_t)0x00000000)
00384 #define RCC_MC02Source_PLLI2SCLK          ((uint32_t)0x40000000)
00385 #define RCC_MC02Source_HSE                 ((uint32_t)0x80000000)
00386 #define RCC_MC02Source_PLLCLK              ((uint32_t)0xc0000000)
00387 #define RCC_MC02Div_1                      ((uint32_t)0x00000000)
00388 #define RCC_MC02Div_2                      ((uint32_t)0x20000000)
00389 #define RCC_MC02Div_3                      ((uint32_t)0x28000000)
00390 #define RCC_MC02Div_4                      ((uint32_t)0x30000000)
00391 #define RCC_MC02Div_5                      ((uint32_t)0x38000000)
00392 #define IS_RCC_MC02SOURCE(SOURCE) (((SOURCE) == RCC_MC02Source_SYSCLK) || ((SOURCE) ==
RCC_MC02Source_PLLI2SCLK) || \
00393                                     ((SOURCE) == RCC_MC02Source_HSE) || ((SOURCE) ==
RCC_MC02Source_PLLCLK))
00394
00395 #define IS_RCC_MC02DIV(DIV) (((DIV) == RCC_MC02Div_1) || ((DIV) == RCC_MC02Div_2) || \
00396                             ((DIV) == RCC_MC02Div_3) || ((DIV) == RCC_MC02Div_4) || \
00397                             ((DIV) == RCC_MC02Div_5))
00405 #define RCC_FLAG_HSIRDY                   ((uint8_t)0x21)
00406 #define RCC_FLAG_HSERDY                   ((uint8_t)0x31)
00407 #define RCC_FLAG_PLLRDY                   ((uint8_t)0x39)
00408 #define RCC_FLAG_PLLI2SRDY               ((uint8_t)0x3B)
00409 #define RCC_FLAG_LSERDY                   ((uint8_t)0x41)
00410 #define RCC_FLAG_LSIRDY                   ((uint8_t)0x61)
00411 #define RCC_FLAG_BORRST                   ((uint8_t)0x79)
00412 #define RCC_FLAG_PINRST                   ((uint8_t)0x7A)
00413 #define RCC_FLAG_PORRST                   ((uint8_t)0x7B)
00414 #define RCC_FLAG_SFTRST                   ((uint8_t)0x7C)
00415 #define RCC_FLAG_IWDGRST                  ((uint8_t)0x7D)
00416 #define RCC_FLAG_WWDGRST                  ((uint8_t)0x7E)
00417 #define RCC_FLAG_LPWRRST                  ((uint8_t)0x7F)
00418 #define IS_RCC_FLAG(FLAG) (((FLAG) == RCC_FLAG_HSIRDY) || ((FLAG) == RCC_FLAG_HSERDY) || \
00419                             ((FLAG) == RCC_FLAG_PLLRDY) || ((FLAG) == RCC_FLAG_LSERDY) || \
00420                             ((FLAG) == RCC_FLAG_LSIRDY) || ((FLAG) == RCC_FLAG_BORRST) || \
00421                             ((FLAG) == RCC_FLAG_PINRST) || ((FLAG) == RCC_FLAG_PORRST) || \

```

```

00422             ((FLAG) == RCC_FLAG_SFTRST) || ((FLAG) == RCC_FLAG_IWDGRST) || \
00423             ((FLAG) == RCC_FLAG_WWDGRST) || ((FLAG) == RCC_FLAG_LPWRRST) || \
00424             ((FLAG) == RCC_FLAG_PLI2SRDY))
00425 #define IS_RCC_CALIBRATION_VALUE(VALUE) ((VALUE) <= 0x1F)
00434 /* Exported macro -----*/
00435 /* Exported functions -----*/
00436
00437 /* Function used to set the RCC clock configuration to the default reset state */
00438 void RCC_DeInit(void);
00439
00440 /* Internal/external clocks, PLL, CSS and MCO configuration functions -----*/
00441 void RCC_HSEConfig(uint8_t RCC_HSE);
00442 ErrorStatus RCC_WaitForHSEStartUp(void);
00443 void RCC_AdjustHSICalibrationValue(uint8_t HSICalibrationValue);
00444 void RCC_HSICmd(FunctionalState NewState);
00445 void RCC_LSEConfig(uint8_t RCC_LSE);
00446 void RCC_LSICmd(FunctionalState NewState);
00447
00448 void RCC_PLLConfig(uint32_t RCC_PLLSource, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP, uint32_t
    PLLQ);
00449 void RCC_PLLCmd(FunctionalState NewState);
00450 void RCC_PLLI2SConfig(uint32_t PLLI2SN, uint32_t PLLI2SR);
00451 void RCC_PLLI2SCmd(FunctionalState NewState);
00452
00453 void RCC_ClockSecuritySystemCmd(FunctionalState NewState);
00454 void RCC_MCO1Config(uint32_t RCC_MCO1Source, uint32_t RCC_MCO1Div);
00455 void RCC_MCO2Config(uint32_t RCC_MCO2Source, uint32_t RCC_MCO2Div);
00456
00457 /* System, AHB and APB busses clocks configuration functions -----*/
00458 void RCC_SYSCLKConfig(uint32_t RCC_SYSCLKSource);
00459 uint8_t RCC_GetSYSCLKSource(void);
00460 void RCC_HCLKConfig(uint32_t RCC_SYSCLK);
00461 void RCC_PCLK1Config(uint32_t RCC_HCLK);
00462 void RCC_PCLK2Config(uint32_t RCC_HCLK);
00463 void RCC_GetClocksFreq(RCC_ClocksTypeDef* RCC_Clocks);
00464
00465 /* Peripheral clocks configuration functions -----*/
00466 void RCC_RTCCLKConfig(uint32_t RCC_RTCCLKSource);
00467 void RCC_RTCCLKCmd(FunctionalState NewState);
00468 void RCC_BackupResetCmd(FunctionalState NewState);
00469 void RCC_I2SCLKConfig(uint32_t RCC_I2SCLKSource);
00470
00471 void RCC_AHB1PeriphClockCmd(uint32_t RCC_AHB1Periph, FunctionalState NewState);
00472 void RCC_AHB2PeriphClockCmd(uint32_t RCC_AHB2Periph, FunctionalState NewState);
00473 void RCC_AHB3PeriphClockCmd(uint32_t RCC_AHB3Periph, FunctionalState NewState);
00474 void RCC_APB1PeriphClockCmd(uint32_t RCC_APB1Periph, FunctionalState NewState);
00475 void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState NewState);
00476
00477 void RCC_AHB1PeriphResetCmd(uint32_t RCC_AHB1Periph, FunctionalState NewState);
00478 void RCC_AHB2PeriphResetCmd(uint32_t RCC_AHB2Periph, FunctionalState NewState);
00479 void RCC_AHB3PeriphResetCmd(uint32_t RCC_AHB3Periph, FunctionalState NewState);
00480 void RCC_APB1PeriphResetCmd(uint32_t RCC_APB1Periph, FunctionalState NewState);
00481 void RCC_APB2PeriphResetCmd(uint32_t RCC_APB2Periph, FunctionalState NewState);
00482
00483 void RCC_AHB1PeriphClockLPModeCmd(uint32_t RCC_AHB1Periph, FunctionalState NewState);
00484 void RCC_AHB2PeriphClockLPModeCmd(uint32_t RCC_AHB2Periph, FunctionalState NewState);
00485 void RCC_AHB3PeriphClockLPModeCmd(uint32_t RCC_AHB3Periph, FunctionalState NewState);
00486 void RCC_APB1PeriphClockLPModeCmd(uint32_t RCC_APB1Periph, FunctionalState NewState);
00487 void RCC_APB2PeriphClockLPModeCmd(uint32_t RCC_APB2Periph, FunctionalState NewState);
00488
00489 /* Interrupts and flags management functions -----*/
00490 void RCC_ITConfig(uint8_t RCC_IT, FunctionalState NewState);
00491 FlagStatus RCC_GetFlagStatus(uint8_t RCC_FLAG);
00492 void RCC_ClearFlag(void);
00493 ITStatus RCC_GetITStatus(uint8_t RCC_IT);
00494 void RCC_ClearITPendingBit(uint8_t RCC_IT);
00495
00496 #ifdef __cplusplus
00497 }
00498 #endif
00499
00500 #endif /* __STM32F4xx_RCC_H */
00501
00510 /***** (C) COPYRIGHT 2011 STMicroelectronics *****END OF FILE*****/

```

6.27 drivers/stm32f4xx_usart.c File Reference

This file provides firmware functions to manage the following functionalities of the Universal synchronous asynchronous receiver transmitter (USART):

```

#include "stm32f4xx_rcc.h"
#include "stm32f4xx_usart.h"

```

Macros

- `#define CR1_CLEAR_MASK`
- `#define CR2_CLOCK_CLEAR_MASK`
- `#define CR3_CLEAR_MASK ((uint16_t)(USART_CR3_RTSE | USART_CR3_CTSE))`
- `#define IT_MASK ((uint16_t)0x001F)`

Functions

- void `USART_DeInit` (USART_TypeDef *USARTx)
Deinitializes the USARTx peripheral registers to their default reset values.
- void `USART_Init` (USART_TypeDef *USARTx, USART_InitTypeDef *USART_InitStruct)
Initializes the USARTx peripheral according to the specified parameters in the USART_InitStruct.
- void `USART_StructInit` (USART_InitTypeDef *USART_InitStruct)
Fills each USART_InitStruct member with its default value.
- void `USART_ClockInit` (USART_TypeDef *USARTx, USART_ClockInitTypeDef *USART_ClockInitStruct)
Initializes the USARTx peripheral Clock according to the specified parameters in the USART_ClockInitStruct.
- void `USART_ClockStructInit` (USART_ClockInitTypeDef *USART_ClockInitStruct)
Fills each USART_ClockInitStruct member with its default value.
- void `USART_Cmd` (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the specified USART peripheral.
- void `USART_SetPrescaler` (USART_TypeDef *USARTx, uint8_t USART_Prescaler)
Sets the system clock prescaler.
- void `USART_OverSampling8Cmd` (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's 8x oversampling mode.
- void `USART_OneBitMethodCmd` (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's one bit sampling method.
- void `USART_SendData` (USART_TypeDef *USARTx, uint16_t Data)
Transmits single data through the USARTx peripheral.
- uint16_t `USART_ReceiveData` (USART_TypeDef *USARTx)
Returns the most recent received data by the USARTx peripheral.
- void `USART_SetAddress` (USART_TypeDef *USARTx, uint8_t USART_Address)
Sets the address of the USART node.
- void `USART_ReceiverWakeUpCmd` (USART_TypeDef *USARTx, FunctionalState NewState)
Determines if the USART is in mute mode or not.
- void `USART_WakeUpConfig` (USART_TypeDef *USARTx, uint16_t USART_WakeUp)
Selects the USART WakeUp method.
- void `USART_LINBreakDetectLengthConfig` (USART_TypeDef *USARTx, uint16_t USART_LINBreakDetectLength)
Sets the USART LIN Break detection length.
- void `USART_LINCmd` (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's LIN mode.
- void `USART_SendBreak` (USART_TypeDef *USARTx)
Transmits break characters.
- void `USART_HalfDuplexCmd` (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's Half Duplex communication.
- void `USART_SetGuardTime` (USART_TypeDef *USARTx, uint8_t USART_GuardTime)
Sets the specified USART guard time.
- void `USART_SmartCardCmd` (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's Smart Card mode.

- void [USART_SmartCardNACKCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables NACK transmission.
- void [USART_IrDAConfig](#) (USART_TypeDef *USARTx, uint16_t USART_IrDAMode)
Configures the USART's IrDA interface.
- void [USART_IrDACmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's IrDA interface.
- void [USART_DMAMCmd](#) (USART_TypeDef *USARTx, uint16_t USART_DMAREq, FunctionalState NewState)
Enables or disables the USART's DMA interface.
- void [USART_ITConfig](#) (USART_TypeDef *USARTx, uint16_t USART_IT, FunctionalState NewState)
Enables or disables the specified USART interrupts.
- FlagStatus [USART_GetFlagStatus](#) (USART_TypeDef *USARTx, uint16_t USART_FLAG)
Checks whether the specified USART flag is set or not.
- void [USART_ClearFlag](#) (USART_TypeDef *USARTx, uint16_t USART_FLAG)
Clears the USARTx's pending flags.
- ITStatus [USART_GetITStatus](#) (USART_TypeDef *USARTx, uint16_t USART_IT)
Checks whether the specified USART interrupt has occurred or not.
- void [USART_ClearITPendingBit](#) (USART_TypeDef *USARTx, uint16_t USART_IT)
Clears the USARTx's interrupt pending bits.

6.27.1 Detailed Description

This file provides firmware functions to manage the following functionalities of the Universal synchronous asynchronous receiver transmitter (USART):

Author

MCD Application Team

Version

V1.0.0

Date

30-September-2011

- Initialization and Configuration
- Data transfers
- Multi-Processor Communication
- LIN mode
- Half-duplex mode
- Smartcard mode
- IrDA mode
- DMA transfers management
- Interrupts and flags management

```
*
*  =====
*                      How to use this driver
*  =====
*
*  1. Enable peripheral clock using the following functions
*      RCC_APB2PeriphClockCmd(RCC_APB2Periph_USARTx, ENABLE) for USART1 and USART6
*      RCC_APB1PeriphClockCmd(RCC_APB1Periph_USARTx, ENABLE) for USART2, USART3, UART4 or UART5.
*
*  2. According to the USART mode, enable the GPIO clocks using
*      RCC_AHB1PeriphClockCmd() function. (The I/O can be TX, RX, CTS,
*      or/and SCLK).
```



```

*
*      3. Peripheral's alternate function:
*          - Connect the pin to the desired peripherals' Alternate
*            Function (AF) using GPIO_PinAFConfig() function
*          - Configure the desired pin in alternate function by:
*            GPIO_InitStruct->GPIO_Mode = GPIO_Mode_AF
*          - Select the type, pull-up/pull-down and output speed via
*            GPIO_PuPd, GPIO_OType and GPIO_Speed members
*          - Call GPIO_Init() function
*
*      4. Program the Baud Rate, Word Length , Stop Bit, Parity, Hardware
*          flow control and Mode(Receiver/Transmitter) using the USART_Init()
*          function.
*
*      5. For synchronous mode, enable the clock and program the polarity,
*          phase and last bit using the USART_ClockInit() function.
*
*      5. Enable the NVIC and the corresponding interrupt using the function
*          USART_ITConfig() if you need to use interrupt mode.
*
*      6. When using the DMA mode
*          - Configure the DMA using DMA_Init() function
*          - Active the needed channel Request using USART_DMAMCmd() function
*
*      7. Enable the USART using the USART_Cmd() function.
*
*      8. Enable the DMA using the DMA_Cmd() function, when using DMA mode.
*
*      Refer to Multi-Processor, LIN, half-duplex, Smartcard, IrDA sub-sections
*      for more details
*
*      In order to reach higher communication baudrates, it is possible to
*      enable the oversampling by 8 mode using the function USART_OverSampling8Cmd().
*      This function should be called after enabling the USART clock (RCC_APBxPeriphClockCmd())
*      and before calling the function USART_Init().
*
*

```

Attention

THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

© COPYRIGHT 2011 STMicroelectronics

6.28 drivers/stm32f4xx_usart.h File Reference

This file contains all the functions prototypes for the USART firmware library.

```
#include "stm32f4xx.h"
```

Data Structures

- struct [USART_InitTypeDef](#)
USART Init Structure definition
- struct [USART_ClockInitTypeDef](#)
USART Clock Init Structure definition

Macros

- `#define IS_USART_ALL_PERIPH(PERIPH)`
- `#define IS_USART_1236_PERIPH(PERIPH)`
- `#define USART_WordLength_8b ((uint16_t)0x0000)`
- `#define USART_WordLength_9b ((uint16_t)0x1000)`
- `#define IS_USART_WORD_LENGTH(LENGTH)`
- `#define USART_StopBits_1 ((uint16_t)0x0000)`
- `#define USART_StopBits_0_5 ((uint16_t)0x1000)`
- `#define USART_StopBits_2 ((uint16_t)0x2000)`
- `#define USART_StopBits_1_5 ((uint16_t)0x3000)`
- `#define IS_USART_STOPBITS(STOPBITS)`
- `#define USART_Parity_No ((uint16_t)0x0000)`
- `#define USART_Parity_Even ((uint16_t)0x0400)`
- `#define USART_Parity_Odd ((uint16_t)0x0600)`
- `#define IS_USART_PARITY(PARITY)`
- `#define USART_Mode_Rx ((uint16_t)0x0004)`
- `#define USART_Mode_Tx ((uint16_t)0x0008)`
- `#define IS_USART_MODE(MODE) (((MODE) & (uint16_t)0xFFFF3) == 0x00) && ((MODE) != (uint16_t)0x00)`
- `#define USART_HardwareFlowControl_None ((uint16_t)0x0000)`
- `#define USART_HardwareFlowControl_RTS ((uint16_t)0x0100)`
- `#define USART_HardwareFlowControl_CTS ((uint16_t)0x0200)`
- `#define USART_HardwareFlowControl_RTS_CTS ((uint16_t)0x0300)`
- `#define IS_USART_HARDWARE_FLOW_CONTROL(CONTROL)`
- `#define USART_Clock_Disable ((uint16_t)0x0000)`
- `#define USART_Clock_Enable ((uint16_t)0x0800)`
- `#define IS_USART_CLOCK(CLOCK)`
- `#define USART_CPOL_Low ((uint16_t)0x0000)`
- `#define USART_CPOL_High ((uint16_t)0x0400)`
- `#define IS_USART_CPOL(CPOL) (((CPOL) == USART_CPOL_Low) || ((CPOL) == USART_CPOL_High))`
- `#define USART_CPHA_1Edge ((uint16_t)0x0000)`
- `#define USART_CPHA_2Edge ((uint16_t)0x0200)`
- `#define IS_USART_CPHA(CPHA) (((CPHA) == USART_CPHA_1Edge) || ((CPHA) == USART_CPHA_2Edge))`
- `#define USART_LastBit_Disable ((uint16_t)0x0000)`
- `#define USART_LastBit_Enable ((uint16_t)0x0100)`
- `#define IS_USART_LASTBIT(LASTBIT)`
- `#define USART_IT_PE ((uint16_t)0x0028)`
- `#define USART_IT_TXE ((uint16_t)0x0727)`
- `#define USART_IT_TC ((uint16_t)0x0626)`
- `#define USART_IT_RXNE ((uint16_t)0x0525)`
- `#define USART_IT_ORE_RX ((uint16_t)0x0325) /* In case interrupt is generated if the RXNEIE bit is set */`
- `#define USART_IT_IDLE ((uint16_t)0x0424)`
- `#define USART_IT_LBD ((uint16_t)0x0846)`
- `#define USART_IT_CTS ((uint16_t)0x096A)`
- `#define USART_IT_ERR ((uint16_t)0x0060)`
- `#define USART_IT_ORE_ER ((uint16_t)0x0360) /* In case interrupt is generated if the EIE bit is set */`
- `#define USART_IT_NE ((uint16_t)0x0260)`
- `#define USART_IT_FE ((uint16_t)0x0160)`
- `#define USART_IT_ORE USART_IT_ORE_ER`
- `#define IS_USART_CONFIG_IT(IT)`
- `#define IS_USART_GET_IT(IT)`
- `#define IS_USART_CLEAR_IT(IT)`
- `#define USART_DMARReq_Tx ((uint16_t)0x0080)`
- `#define USART_DMARReq_Rx ((uint16_t)0x0040)`

- `#define IS_USART_DMAREQ(DMAREQ) (((DMAREQ) & (uint16_t)0xFF3F) == 0x00) && ((DMAREQ) != (uint16_t)0x00)`
- `#define USART_WakeUp_IdleLine ((uint16_t)0x0000)`
- `#define USART_WakeUp_AddressMark ((uint16_t)0x0800)`
- `#define IS_USART_WAKEUP(WAKEUP)`
- `#define USART_LINBreakDetectLength_10b ((uint16_t)0x0000)`
- `#define USART_LINBreakDetectLength_11b ((uint16_t)0x0020)`
- `#define IS_USART_LIN_BREAK_DETECT_LENGTH(LENGTH)`
- `#define USART_IrDAMode_LowPower ((uint16_t)0x0004)`
- `#define USART_IrDAMode_Normal ((uint16_t)0x0000)`
- `#define IS_USART_IRDA_MODE(MODE)`
- `#define USART_FLAG_CTS ((uint16_t)0x0200)`
- `#define USART_FLAG_LBD ((uint16_t)0x0100)`
- `#define USART_FLAG_TXE ((uint16_t)0x0080)`
- `#define USART_FLAG_TC ((uint16_t)0x0040)`
- `#define USART_FLAG_RXNE ((uint16_t)0x0020)`
- `#define USART_FLAG_IDLE ((uint16_t)0x0010)`
- `#define USART_FLAG_ORE ((uint16_t)0x0008)`
- `#define USART_FLAG_NE ((uint16_t)0x0004)`
- `#define USART_FLAG_FE ((uint16_t)0x0002)`
- `#define USART_FLAG_PE ((uint16_t)0x0001)`
- `#define IS_USART_FLAG(FLAG)`
- `#define IS_USART_CLEAR_FLAG(FLAG) (((FLAG) & (uint16_t)0xFC9F) == 0x00) && ((FLAG) != (uint16_t)0x00)`
- `#define IS_USART_BAUDRATE(BAUDRATE) (((BAUDRATE) > 0) && ((BAUDRATE) < 7500001))`
- `#define IS_USART_ADDRESS(ADDRESS) ((ADDRESS) <= 0xF)`
- `#define IS_USART_DATA(DATA) ((DATA) <= 0x1FF)`

Functions

- void `USART_DeInit` (USART_TypeDef *USARTx)
Deinitializes the USARTx peripheral registers to their default reset values.
- void `USART_Init` (USART_TypeDef *USARTx, USART_InitTypeDef *USART_InitStruct)
Initializes the USARTx peripheral according to the specified parameters in the USART_InitStruct .
- void `USART_StructInit` (USART_InitTypeDef *USART_InitStruct)
Fills each USART_InitStruct member with its default value.
- void `USART_ClockInit` (USART_TypeDef *USARTx, USART_ClockInitTypeDef *USART_ClockInitStruct)
Initializes the USARTx peripheral Clock according to the specified parameters in the USART_ClockInitStruct .
- void `USART_ClockStructInit` (USART_ClockInitTypeDef *USART_ClockInitStruct)
Fills each USART_ClockInitStruct member with its default value.
- void `USART_Cmd` (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the specified USART peripheral.
- void `USART_SetPrescaler` (USART_TypeDef *USARTx, uint8_t USART_Prescaler)
Sets the system clock prescaler.
- void `USART_OverSampling8Cmd` (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's 8x oversampling mode.
- void `USART_OneBitMethodCmd` (USART_TypeDef *USARTx, FunctionalState NewState)
Enables or disables the USART's one bit sampling method.
- void `USART_SendData` (USART_TypeDef *USARTx, uint16_t Data)
Transmits single data through the USARTx peripheral.
- uint16_t `USART_ReceiveData` (USART_TypeDef *USARTx)
Returns the most recent received data by the USARTx peripheral.
- void `USART_SetAddress` (USART_TypeDef *USARTx, uint8_t USART_Address)

- Sets the address of the USART node.*

 - void [USART_WakeUpConfig](#) (USART_TypeDef *USARTx, uint16_t USART_WakeUp)

Selects the USART WakeUp method.
- void [USART_ReceiverWakeUpCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)

Determines if the USART is in mute mode or not.
- void [USART_LINBreakDetectLengthConfig](#) (USART_TypeDef *USARTx, uint16_t USART_LINBreakDetectLength)

Sets the USART LIN Break detection length.
- void [USART_LINCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)

Enables or disables the USART's LIN mode.
- void [USART_SendBreak](#) (USART_TypeDef *USARTx)

Transmits break characters.
- void [USART_HalfDuplexCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)

Enables or disables the USART's Half Duplex communication.
- void [USART_SmartCardCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)

Enables or disables the USART's Smart Card mode.
- void [USART_SmartCardNACKCmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)

Enables or disables NACK transmission.
- void [USART_SetGuardTime](#) (USART_TypeDef *USARTx, uint8_t USART_GuardTime)

Sets the specified USART guard time.
- void [USART_IrDAConfig](#) (USART_TypeDef *USARTx, uint16_t USART_IrDAMode)

Configures the USART's IrDA interface.
- void [USART_IrDACmd](#) (USART_TypeDef *USARTx, FunctionalState NewState)

Enables or disables the USART's IrDA interface.
- void [USART_DMAMCmd](#) (USART_TypeDef *USARTx, uint16_t USART_DMAREq, FunctionalState NewState)

Enables or disables the USART's DMA interface.
- void [USART_ITConfig](#) (USART_TypeDef *USARTx, uint16_t USART_IT, FunctionalState NewState)

Enables or disables the specified USART interrupts.
- FlagStatus [USART_GetFlagStatus](#) (USART_TypeDef *USARTx, uint16_t USART_FLAG)

Checks whether the specified USART flag is set or not.
- void [USART_ClearFlag](#) (USART_TypeDef *USARTx, uint16_t USART_FLAG)

Clears the USARTx's pending flags.
- ITStatus [USART_GetITStatus](#) (USART_TypeDef *USARTx, uint16_t USART_IT)

Checks whether the specified USART interrupt has occurred or not.
- void [USART_ClearITPendingBit](#) (USART_TypeDef *USARTx, uint16_t USART_IT)

Clears the USARTx's interrupt pending bits.

6.28.1 Detailed Description

This file contains all the functions prototypes for the USART firmware library.

Author

MCD Application Team

Version

V1.0.0

Date

30-September-2011

Attention

THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.

© COPYRIGHT 2011 STMicroelectronics

6.29 stm32f4xx_usart.h

[Go to the documentation of this file.](#)

```

00001
00023 /* Define to prevent recursive inclusion -----*/
00024 #ifndef __STM32F4xx_USART_H
00025 #define __STM32F4xx_USART_H
00026
00027 #ifdef __cplusplus
00028     extern "C" {
00029 #endif
00030
00031 /* Includes -----*/
00032 #include "stm32f4xx.h"
00033
00042 /* Exported types -----*/
00043
00048 typedef struct
00049 {
00050     uint32_t  USART_BaudRate;
00056     uint16_t  USART_WordLength;
00059     uint16_t  USART_StopBits;
00062     uint16_t  USART_Parity;
00069     uint16_t  USART_Mode;
00072     uint16_t  USART_HardwareFlowControl;
00075 } USART_InitTypeDef;
00076
00081 typedef struct
00082 {
00083
00084     uint16_t  USART_Clock;
00087     uint16_t  USART_CPOL;
00090     uint16_t  USART_CPHA;
00093     uint16_t  USART_LastBit;
00096 } USART_ClockInitTypeDef;
00097
00098 /* Exported constants -----*/
00099
00104 #define IS_USART_ALL_PERIPH(PERIPH) (((PERIPH) == USART1) || \
00105                                     ((PERIPH) == USART2) || \
00106                                     ((PERIPH) == USART3) || \
00107                                     ((PERIPH) == UART4) || \
00108                                     ((PERIPH) == UART5) || \
00109                                     ((PERIPH) == USART6))
00110
00111 #define IS_USART_1236_PERIPH(PERIPH) (((PERIPH) == USART1) || \
00112                                     ((PERIPH) == USART2) || \
00113                                     ((PERIPH) == USART3) || \
00114                                     ((PERIPH) == USART6))
00115
00120 #define USART_WordLength_8b                ((uint16_t)0x0000)
00121 #define USART_WordLength_9b                ((uint16_t)0x1000)
00122
00123 #define IS_USART_WORD_LENGTH(LENGTH) (((LENGTH) == USART_WordLength_8b) || \
00124                                     ((LENGTH) == USART_WordLength_9b))
00133 #define USART_StopBits_1                    ((uint16_t)0x0000)
00134 #define USART_StopBits_0_5                  ((uint16_t)0x1000)
00135 #define USART_StopBits_2                    ((uint16_t)0x2000)
00136 #define USART_StopBits_1_5                  ((uint16_t)0x3000)
00137 #define IS_USART_STOPBITS(STOPBITS) (((STOPBITS) == USART_StopBits_1) || \
00138                                     ((STOPBITS) == USART_StopBits_0_5) || \
00139                                     ((STOPBITS) == USART_StopBits_2) || \
00140                                     ((STOPBITS) == USART_StopBits_1_5))
00149 #define USART_Parity_No                     ((uint16_t)0x0000)
00150 #define USART_Parity_Even                   ((uint16_t)0x0400)

```

```

00151 #define USART_Parity_Odd ((uint16_t)0x0600)
00152 #define IS_USART_PARITY(PARITY) (((PARITY) == USART_Parity_No) || \
00153 ((PARITY) == USART_Parity_Even) || \
00154 ((PARITY) == USART_Parity_Odd))
00163 #define USART_Mode_Rx ((uint16_t)0x0004)
00164 #define USART_Mode_Tx ((uint16_t)0x0008)
00165 #define IS_USART_MODE(MODE) (((MODE) & (uint16_t)0xFFFF3) == 0x00) && ((MODE) != (uint16_t)0x00)
00173 #define USART_HardwareFlowControl_None ((uint16_t)0x0000)
00174 #define USART_HardwareFlowControl_RTS ((uint16_t)0x0100)
00175 #define USART_HardwareFlowControl_CTS ((uint16_t)0x0200)
00176 #define USART_HardwareFlowControl_RTS_CTS ((uint16_t)0x0300)
00177 #define IS_USART_HARDWARE_FLOW_CONTROL(CONTROL) \
00178 ((CONTROL) == USART_HardwareFlowControl_None) || \
00179 ((CONTROL) == USART_HardwareFlowControl_RTS) || \
00180 ((CONTROL) == USART_HardwareFlowControl_CTS) || \
00181 ((CONTROL) == USART_HardwareFlowControl_RTS_CTS))
00189 #define USART_Clock_Disable ((uint16_t)0x0000)
00190 #define USART_Clock_Enable ((uint16_t)0x0800)
00191 #define IS_USART_CLOCK(CLOCK) ((CLOCK) == USART_Clock_Disable) || \
00192 ((CLOCK) == USART_Clock_Enable))
00201 #define USART_CPOL_Low ((uint16_t)0x0000)
00202 #define USART_CPOL_High ((uint16_t)0x0400)
00203 #define IS_USART_CPOL(CPOL) ((CPOL) == USART_CPOL_Low) || ((CPOL) == USART_CPOL_High))
00204
00213 #define USART_CPHA_1Edge ((uint16_t)0x0000)
00214 #define USART_CPHA_2Edge ((uint16_t)0x0200)
00215 #define IS_USART_CPHA(CPHA) ((CPHA) == USART_CPHA_1Edge) || ((CPHA) == USART_CPHA_2Edge))
00216
00225 #define USART_LastBit_Disable ((uint16_t)0x0000)
00226 #define USART_LastBit_Enable ((uint16_t)0x0100)
00227 #define IS_USART_LASTBIT(LASTBIT) ((LASTBIT) == USART_LastBit_Disable) || \
00228 ((LASTBIT) == USART_LastBit_Enable))
00237 #define USART_IT_PE ((uint16_t)0x0028)
00238 #define USART_IT_TXE ((uint16_t)0x0727)
00239 #define USART_IT_TC ((uint16_t)0x0626)
00240 #define USART_IT_RXNE ((uint16_t)0x0525)
00241 #define USART_IT_ORE_RX ((uint16_t)0x0325) /* In case interrupt is generated if
the RXNEIE bit is set */
00242 #define USART_IT_IDLE ((uint16_t)0x0424)
00243 #define USART_IT_LBD ((uint16_t)0x0846)
00244 #define USART_IT_CTS ((uint16_t)0x096A)
00245 #define USART_IT_ERR ((uint16_t)0x0060)
00246 #define USART_IT_ORE_ER ((uint16_t)0x0360) /* In case interrupt is generated if
the EIE bit is set */
00247 #define USART_IT_NE ((uint16_t)0x0260)
00248 #define USART_IT_FE ((uint16_t)0x0160)
00249
00253 #define USART_IT_ORE USART_IT_ORE_ER
00258 #define IS_USART_CONFIG_IT(IT) ((IT) == USART_IT_PE) || ((IT) == USART_IT_TXE) || \
00259 ((IT) == USART_IT_TC) || ((IT) == USART_IT_RXNE) || \
00260 ((IT) == USART_IT_IDLE) || ((IT) == USART_IT_LBD) || \
00261 ((IT) == USART_IT_CTS) || ((IT) == USART_IT_ERR))
00262 #define IS_USART_GET_IT(IT) ((IT) == USART_IT_PE) || ((IT) == USART_IT_TXE) || \
00263 ((IT) == USART_IT_TC) || ((IT) == USART_IT_RXNE) || \
00264 ((IT) == USART_IT_IDLE) || ((IT) == USART_IT_LBD) || \
00265 ((IT) == USART_IT_CTS) || ((IT) == USART_IT_ORE) || \
00266 ((IT) == USART_IT_ORE_RX) || ((IT) == USART_IT_ORE_ER) || \
00267 ((IT) == USART_IT_NE) || ((IT) == USART_IT_FE))
00268 #define IS_USART_CLEAR_IT(IT) ((IT) == USART_IT_TC) || ((IT) == USART_IT_RXNE) || \
00269 ((IT) == USART_IT_LBD) || ((IT) == USART_IT_CTS))
00278 #define USART_DMAReq_Tx ((uint16_t)0x0080)
00279 #define USART_DMAReq_Rx ((uint16_t)0x0040)
00280 #define IS_USART_DMAREQ(DMAREQ) (((DMAREQ) & (uint16_t)0xFF3F) == 0x00) && ((DMAREQ) !=
(uint16_t)0x00)
00281
00290 #define USART_WakeUp_IdleLine ((uint16_t)0x0000)
00291 #define USART_WakeUp_AddressMark ((uint16_t)0x0800)
00292 #define IS_USART_WAKEUP(WAKEUP) ((WAKEUP) == USART_WakeUp_IdleLine) || \
00293 ((WAKEUP) == USART_WakeUp_AddressMark))
00302 #define USART_LINBreakDetectLength_10b ((uint16_t)0x0000)
00303 #define USART_LINBreakDetectLength_11b ((uint16_t)0x0020)
00304 #define IS_USART_LIN_BREAK_DETECT_LENGTH(LENGTH) \
00305 ((LENGTH) == USART_LINBreakDetectLength_10b) || \
00306 ((LENGTH) == USART_LINBreakDetectLength_11b))
00315 #define USART_IrDAMode_LowPower ((uint16_t)0x0004)
00316 #define USART_IrDAMode_Normal ((uint16_t)0x0000)
00317 #define IS_USART_IRDA_MODE(MODE) ((MODE) == USART_IrDAMode_LowPower) || \
00318 ((MODE) == USART_IrDAMode_Normal))
00327 #define USART_FLAG_CTS ((uint16_t)0x0200)
00328 #define USART_FLAG_LBD ((uint16_t)0x0100)
00329 #define USART_FLAG_TXE ((uint16_t)0x0080)
00330 #define USART_FLAG_TC ((uint16_t)0x0040)
00331 #define USART_FLAG_RXNE ((uint16_t)0x0020)
00332 #define USART_FLAG_IDLE ((uint16_t)0x0010)
00333 #define USART_FLAG_ORE ((uint16_t)0x0008)
00334 #define USART_FLAG_NE ((uint16_t)0x0004)
00335 #define USART_FLAG_FE ((uint16_t)0x0002)

```

```

00336 #define USART_FLAG_PE ((uint16_t)0x0001)
00337 #define IS_USART_FLAG(FLAG) (((FLAG) == USART_FLAG_PE) || ((FLAG) == USART_FLAG_TXE) || \
00338 ((FLAG) == USART_FLAG_TC) || ((FLAG) == USART_FLAG_RXNE) || \
00339 ((FLAG) == USART_FLAG_IDLE) || ((FLAG) == USART_FLAG_LBD) || \
00340 ((FLAG) == USART_FLAG_CTS) || ((FLAG) == USART_FLAG_ORE) || \
00341 ((FLAG) == USART_FLAG_NE) || ((FLAG) == USART_FLAG_FE))
00342
00343 #define IS_USART_CLEAR_FLAG(FLAG) (((FLAG) & (uint16_t)0xFC9F) == 0x00) && ((FLAG) !=
(uint16_t)0x00)
00344
00345 #define IS_USART_BAUDRATE(BAUDRATE) (((BAUDRATE) > 0) && ((BAUDRATE) < 7500001))
00346 #define IS_USART_ADDRESS(ADDRESS) ((ADDRESS) <= 0xF)
00347 #define IS_USART_DATA(DATA) ((DATA) <= 0x1FF)
00348
00357 /* Exported macro -----*/
00358 /* Exported functions -----*/
00359
00360 /* Function used to set the USART configuration to the default reset state */
00361 void USART_DeInit(USART_TypeDef* USARTx);
00362
00363 /* Initialization and Configuration functions -----*/
00364 void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct);
00365 void USART_StructInit(USART_InitTypeDef* USART_InitStruct);
00366 void USART_ClockInit(USART_TypeDef* USARTx, USART_ClockInitTypeDef* USART_ClockInitStruct);
00367 void USART_ClockStructInit(USART_ClockInitTypeDef* USART_ClockInitStruct);
00368 void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState);
00369 void USART_SetPrescaler(USART_TypeDef* USARTx, uint8_t USART_Prescaler);
00370 void USART_OverSampling8Cmd(USART_TypeDef* USARTx, FunctionalState NewState);
00371 void USART_OneBitMethodCmd(USART_TypeDef* USARTx, FunctionalState NewState);
00372
00373 /* Data transfers functions -----*/
00374 void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
00375 uint16_t USART_ReceiveData(USART_TypeDef* USARTx);
00376
00377 /* Multi-Processor Communication functions -----*/
00378 void USART_SetAddress(USART_TypeDef* USARTx, uint8_t USART_Address);
00379 void USART_WakeUpConfig(USART_TypeDef* USARTx, uint16_t USART_WakeUp);
00380 void USART_ReceiverWakeUpCmd(USART_TypeDef* USARTx, FunctionalState NewState);
00381
00382 /* LIN mode functions -----*/
00383 void USART_LINBreakDetectLengthConfig(USART_TypeDef* USARTx, uint16_t USART_LINBreakDetectLength);
00384 void USART_LINCmd(USART_TypeDef* USARTx, FunctionalState NewState);
00385 void USART_SendBreak(USART_TypeDef* USARTx);
00386
00387 /* Half-duplex mode function -----*/
00388 void USART_HalfDuplexCmd(USART_TypeDef* USARTx, FunctionalState NewState);
00389
00390 /* Smartcard mode functions -----*/
00391 void USART_SmartCardCmd(USART_TypeDef* USARTx, FunctionalState NewState);
00392 void USART_SmartCardNACKCmd(USART_TypeDef* USARTx, FunctionalState NewState);
00393 void USART_SetGuardTime(USART_TypeDef* USARTx, uint8_t USART_GuardTime);
00394
00395 /* IrDA mode functions -----*/
00396 void USART_IrDAModeConfig(USART_TypeDef* USARTx, uint16_t USART_IrDAMode);
00397 void USART_IrDACmd(USART_TypeDef* USARTx, FunctionalState NewState);
00398
00399 /* DMA transfers management functions -----*/
00400 void USART_DMAMCmd(USART_TypeDef* USARTx, uint16_t USART_DMAREq, FunctionalState NewState);
00401
00402 /* Interrupts and flags management functions -----*/
00403 void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT, FunctionalState NewState);
00404 FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG);
00405 void USART_ClearFlag(USART_TypeDef* USARTx, uint16_t USART_FLAG);
00406 ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT);
00407 void USART_ClearITPendingBit(USART_TypeDef* USARTx, uint16_t USART_IT);
00408
00409 #ifdef __cplusplus
00410 }
00411 #endif
00412
00413 #endif /* __STM32F4xx_USART_H */
00414
00423 /***** (C) COPYRIGHT 2011 STMicroelectronics *****/

```

6.30 drivers/timer.c File Reference

```

#include <platform.h>
#include <timer.h>

```

Functions

- void [timer_init](#) (uint32_t timestamp)
- void [timer_enable](#) (void)
Enables the timer operation.
- void [timer_disable](#) (void)
Disables the timer.
- void [timer_set_callback](#) (void(*callback)(void))
Pass a callback to the API, which is executed during the interrupt handler.
- void [SysTick_Handler](#) (void)

Variables

- uint32_t [timer_period](#)

6.30.1 Function Documentation

6.30.1.1 SysTick_Handler()

```
void SysTick_Handler (
    void )
```

6.30.1.2 timer_disable()

```
void timer_disable (
    void )
```

Disables the timer.

6.30.1.3 timer_enable()

```
void timer_enable (
    void )
```

Enables the timer operation.

6.30.1.4 timer_init()

```
void timer_init (
    uint32_t timestamp )
```

6.30.1.5 timer_set_callback()

```
void timer_set_callback (
    void(*) (void) callback )
```

Pass a callback to the API, which is executed during the interrupt handler.

Parameters

<i>callback</i>	Callback function.
-----------------	--------------------

6.30.2 Variable Documentation

6.30.2.1 timer_period

```
uint32_t timer_period
```

6.31 drivers/timer.h File Reference

Controller for a hardware timer module.

```
#include <stdint.h>
```

Functions

- void [timer_irq_handler](#) (void)
Initialises the timer with a specified period.
- void [timer_init](#) (uint32_t timestamp)
- void [timer_set_callback](#) (void(*callback)(void))
Pass a callback to the API, which is executed during the interrupt handler.
- void [timer_enable](#) (void)
Enables the timer operation.
- void [timer_disable](#) (void)
Disables the timer.

6.31.1 Detailed Description

Controller for a hardware timer module.

Copyright

ARM University Program © ARM Ltd 2014.

6.31.2 Function Documentation

6.31.2.1 timer_disable()

```
void timer_disable (
    void )
```

Disables the timer.

6.31.2.2 timer_enable()

```
void timer_enable (
    void )
```

Enables the timer operation.

6.31.2.3 timer_init()

```
void timer_init (
    uint32_t timestamp )
```

6.31.2.4 timer_irq_handler()

```
void timer_irq_handler (
    void )
```

Initialises the timer with a specified period.

Parameters

<i>period</i>	Period of the timer tick (in cpu cycles).
---------------	---

6.31.2.5 timer_set_callback()

```
void timer_set_callback (
```

```
void(*) (void) callback )
```

Pass a callback to the API, which is executed during the interrupt handler.

Parameters

<i>callback</i>	Callback function.
-----------------	--------------------

6.32 timer.h

[Go to the documentation of this file.](#)

```
00001
00006 #ifndef TIMER_H
00007 #define TIMER_H
00008 #include <stdint.h>
00009
00014 void timer_irq_handler(void);
00015 void timer_init(uint32_t timestamp);
00016
00021 void timer_set_callback(void (*callback)(void));
00022
00024 void timer_enable(void);
00025
00027 void timer_disable(void);
00028
00029 #endif // TIMER_H
00030
00031 // *****ARM University Program Copyright ARM Ltd
2016*****
```

6.33 drivers/uart.c File Reference

```
#include <platform.h>
#include <uart.h>
#include <STM32F4xx_RCC.h>
#include <STM32F4xx_USART.h>
#include <STM32F4xx_GPIO.h>
```

Functions

- void `uart_init` (uint32_t baud)
Initialises the UART controller.
- void `uart_enable` (void)
Enables UART transmission and reception.
- void `uart_print` (char *string)
Transmit a null terminated string.
- void `uart_set_rx_callback` (void(*callback)(uint8_t))
- void `uart_tx` (uint8_t c)
Transmit a single character.
- uint8_t `uart_rx` (void)
Receive a single character.
- void `USART2_IRQHandler` (void)

6.33.1 Function Documentation

6.33.1.1 `uart_enable()`

```
void uart_enable (
    void )
```

Enables UART transmission and reception.

6.33.1.2 uart_init()

```
void uart_init (
    uint32_t baud )
```

Initialises the UART controller.

Parameters

<i>baud</i>	Baud rate to be used (symbols per second).
-------------	--

6.33.1.3 uart_print()

```
void uart_print (
    char * str )
```

Transmit a null terminated string.

Parameters

<i>str</i>	String to be sent.
------------	--------------------

6.33.1.4 uart_rx()

```
uint8_t uart_rx (
    void )
```

Receive a single character.

Warning

This function blocks until a character is available. For a non-blocking receive, see [uart_set_rx_callback\(\)](#).

Returns

Received character.

6.33.1.5 uart_set_rx_callback()

```
void uart_set_rx_callback (
    void(*) (uint8_t) callback )
```

6.33.1.6 uart_tx()

```
void uart_tx (
    uint8_t c )
```

Transmit a single character.

Parameters

<i>c</i>	Character to be sent.
----------	-----------------------

6.33.1.7 USART2_IRQHandler()

```
void USART2_IRQHandler (
    void )
```

6.34 drivers/uart.h File Reference

Controller for a hardware UART module.

```
#include <stdint.h>
```

Functions

- void `uart_init` (uint32_t baud)
Initialises the UART controller.
- void `uart_enable` (void)
Enables UART transmission and reception.
- void `uart_tx` (uint8_t c)
Transmit a single character.
- uint8_t `uart_rx` (void)
Receive a single character.
- void `uart_print` (char *str)
Transmit a null terminated string.
- void `uart_set_rx_callback` (void(*callback)(uint8_t c))
Passes a callback function to the API which is executed during the receive interrupt handler.

6.34.1 Detailed Description

Controller for a hardware UART module.

Copyright

ARM University Program © ARM Ltd 2014.

6.34.2 Function Documentation

6.34.2.1 `uart_enable()`

```
void uart_enable (
    void )
```

Enables UART transmission and reception.

6.34.2.2 `uart_init()`

```
void uart_init (
    uint32_t baud )
```

Initialises the UART controller.

Parameters

<i>baud</i>	Baud rate to be used (symbols per second).
-------------	--

6.34.2.3 `uart_print()`

```
void uart_print (
    char * str )
```

Transmit a null terminated string.

Parameters

<i>str</i>	String to be sent.
------------	--------------------

6.34.2.4 uart_rx()

```
uint8_t uart_rx (
    void )
```

Receive a single character.

Warning

This function blocks until a character is available. For a non-blocking receive, see [uart_set_rx_callback\(\)](#).

Returns

Received character.

6.34.2.5 uart_set_rx_callback()

```
void uart_set_rx_callback (
    void(*) (uint8_t c) callback )
```

Passes a callback function to the API which is executed during the receive interrupt handler.

Parameters

<i>callback</i>	Callback function.
-----------------	--------------------

6.34.2.6 uart_tx()

```
void uart_tx (
    uint8_t c )
```

Transmit a single character.

Parameters

<i>c</i>	Character to be sent.
----------	-----------------------

6.35 uart.h

[Go to the documentation of this file.](#)

```
00001
00006 #ifndef UART_H
00007 #define UART_H
00008 #include <stdint.h>
00009
00013 void uart_init(uint32_t baud);
00014
00017 void uart_enable(void);
00018
00022 void uart_tx(uint8_t c);
00023
00030 uint8_t uart_rx(void);
00031
00035 void uart_print(char *str);
00036
00041 void uart_set_rx_callback(void (*callback) (uint8_t c));
00042
00043 #endif // UART_H
00044
00045 // *****ARM University Program Copyright © ARM Ltd
    2016*****
```

- 6.36 Objects/adc.d File Reference**
- 6.37 Objects/comparator.d File Reference**
- 6.38 Objects/gpio.d File Reference**
- 6.39 Objects/i2c.d File Reference**
- 6.40 Objects/main.d File Reference**
- 6.41 Objects/startup_stm32f401xe.d File Reference**
- 6.42 Objects/stm32f4xx_adc.d File Reference**
- 6.43 Objects/stm32f4xx_gpio.d File Reference**
- 6.44 Objects/stm32f4xx_i2c.d File Reference**
- 6.45 Objects/stm32f4xx_rcc.d File Reference**
- 6.46 Objects/stm32f4xx_usart.d File Reference**
- 6.47 Objects/system_stm32f4xx.d File Reference**
- 6.48 Objects/timer.d File Reference**
- 6.49 Objects/uart.d File Reference**
- 6.50 RTE/_Target_1/RTE_Components.h File Reference**

Macros

- `#define CMSIS_device_header "stm32f4xx.h"`
- `#define RTE_DEVICE_STARTUP_STM32F4XX /* Device Startup for STM32F4 */`

6.50.1 Macro Definition Documentation

6.50.1.1 CMSIS_device_header

```
#define CMSIS_device_header "stm32f4xx.h"
```

6.50.1.2 RTE_DEVICE_STARTUP_STM32F4XX

```
#define RTE_DEVICE_STARTUP_STM32F4XX /* Device Startup for STM32F4 */
```

6.51 RTE_Components.h

[Go to the documentation of this file.](#)

```
00001
00002 /*
00003  * Auto generated Run-Time-Environment Configuration File
00004  *      *** Do not modify ! ***
00005  *
00006  * Project: 'AsmProject'
00007  * Target: 'Target 1'
00008  */
```

```

00009
00010 #ifndef RTE_COMPONENTS_H
00011 #define RTE_COMPONENTS_H
00012
00013
00014 /*
00015  * Define the Device Header File:
00016  */
00017 #define CMSIS_device_header "stm32f4xx.h"
00018
00019 /* Keil::Device:Startup:2.6.3 */
00020 #define RTE_DEVICE_STARTUP_STM32F4XX /* Device Startup for STM32F4 */
00021
00022
00023 #endif /* RTE_COMPONENTS_H */

```

6.52 RTE/Device/STM32F401RETx/system_stm32f4xx.c File Reference

CMSIS Cortex-M4 Device Peripheral Access Layer System Source File.

```
#include "stm32f4xx.h"
```

Macros

- #define [HSE_VALUE](#) ((uint32_t)25000000)
- #define [HSI_VALUE](#) ((uint32_t)16000000)

Functions

- void [SystemInit](#) (void)
Setup the microcontroller system Initialize the FPU setting, vector table location and External memory configuration.
- void [SystemCoreClockUpdate](#) (void)
Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Variables

- uint32_t [SystemCoreClock](#) = 16000000
- const uint8_t [AHBPrescTable](#) [16] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
- const uint8_t [APBPrescTable](#) [8] = {0, 0, 0, 0, 1, 2, 3, 4}

6.52.1 Detailed Description

CMSIS Cortex-M4 Device Peripheral Access Layer System Source File.

Author

MCD Application Team

This file provides two functions and one global variable to be called from user application:

- [SystemInit\(\)](#): This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32f4xx.s" file.
- [SystemCoreClock](#) variable: Contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.
- [SystemCoreClockUpdate\(\)](#): Updates the variable [SystemCoreClock](#) and must be called whenever the core clock is changed during program execution.

Attention

© Copyright (c) 2017 STMicroelectronics. All rights reserved.

This software component is licensed by ST under BSD 3-Clause license, the "License"; You may not use this file except in compliance with the License. You may obtain a copy of the License at: opensource.org/licenses/BSD-3-Clause

6.53 src/main.c File Reference

Square Root Approximation using Integer Approach - Project 1 Module 1.

```
#include <string.h>
#include <assert.h>
```

Functions

- `__asm int my_sqrt_int (int x)`
Compute the integer square root of a number using approximate bisection method.
- `__asm int my_sqrt_fixed_point (int x)`
Compute the square root of a number using approximate bisection method in Q16.16 number format.
- `int main (void)`
Application Entry Point.

6.53.1 Detailed Description

Square Root Approximation using Integer Approach - Project 1 Module 1.

Author

Viraj Patel, Kiran jojare

See also

[ARM Cortex-M4 Instructions](#)

PDF obtained from professor "Approximate Square Root Bisection Method"

6.53.2 Function Documentation

6.53.2.1 main()

```
int main (
    void )
```

Application Entry Point.

Parameters

<code>void</code>	
-------------------	--

Returns

nothing

6.53.2.2 my_sqrt_fixed_point()

```
__asm int my_sqrt_fixed_point (
    int x )
```

Compute the square root of a number using approximate bisection method in Q16.16 number format.

Parameters

<i>int</i>	x - The input integer to take the square root in Q16.16 number format.
------------	--

Returns

The square root of the given number in Q16.16 format

6.53.2.3 my_sqrt_int()

```
__asm int my_sqrt_int (  
    int x )
```

Compute the integer square root of a number using approximate bisection method.

Parameters

<i>int</i>	x - The input integer to take the square root.
------------	--

Returns

The integer square root of the given number.

Index

- [_ADC_ConfigChannel](#)
 - [adc.c, 323](#)
 - [adc.h, 334](#)
- [_ADC_GET_FLAG](#)
 - [adc.h, 328](#)
- [_ADC_GetValue](#)
 - [adc.c, 323](#)
 - [adc.h, 334](#)
- [_ADC_Init](#)
 - [adc.c, 324](#)
 - [adc.h, 334](#)
- [_ADC_InitTypeDef, 311](#)
 - [ClockPrescaler, 311](#)
 - [ContinuousConvMode, 311](#)
 - [DataAlign, 311](#)
 - [DiscontinuousConvMode, 311](#)
 - [DMAContinuousRequests, 311](#)
 - [EOCSelection, 312](#)
 - [ExternalTrigConv, 312](#)
 - [ExternalTrigConvEdge, 312](#)
 - [NbrOfConversion, 312](#)
 - [NbrOfDiscConversion, 312](#)
 - [Resolution, 312](#)
 - [ScanConvMode, 312](#)
- [_ADC_PollForConversion](#)
 - [adc.c, 324](#)
 - [adc.h, 334](#)
- [_ADC_Start](#)
 - [adc.c, 324](#)
 - [adc.h, 335](#)
- [_GPIO_Init](#)
 - [adc.c, 324](#)
 - [adc.h, 335](#)
- [_IS_BIT_CLR](#)
 - [adc.h, 328](#)
- [_IS_BIT_SET](#)
 - [adc.h, 328](#)
- [_adc_read](#)
 - [adc.c, 324](#)
 - [adc.h, 335](#)
- [ADC, 9](#)
 - [ADC_AnalogWatchdogCmd, 14](#)
 - [ADC_AnalogWatchdogSingleChannelConfig, 14](#)
 - [ADC_AnalogWatchdogThresholdsConfig, 15](#)
 - [ADC_AutoInjectedConvCmd, 16](#)
 - [ADC_ClearFlag, 16](#)
 - [ADC_ClearITPendingBit, 17](#)
 - [ADC_Cmd, 17](#)
 - [ADC_CommonInit, 17](#)
 - [ADC_CommonStructInit, 18](#)
 - [ADC_ContinuousModeCmd, 18](#)
 - [ADC_DeInit, 19](#)
 - [ADC_DiscModeChannelCountConfig, 19](#)
 - [ADC_DiscModeCmd, 19](#)
 - [ADC_DMACmd, 20](#)
 - [ADC_DMAResquestAfterLastTransferCmd, 20](#)
 - [ADC_EOCOnEachRegularChannelCmd, 20](#)
 - [ADC_ExternalTrigInjectedConvConfig, 21](#)
 - [ADC_ExternalTrigInjectedConvEdgeConfig, 22](#)
 - [ADC_GetConversionValue, 23](#)
 - [ADC_GetFlagStatus, 23](#)
 - [ADC_GetInjectedConversionValue, 24](#)
 - [ADC_GetITStatus, 24](#)
 - [ADC_GetMultiModeConversionValue, 25](#)
 - [ADC_GetSoftwareStartConvStatus, 25](#)
 - [ADC_GetSoftwareStartInjectedConvCmdStatus, 26](#)
 - [ADC_Init, 26](#)
 - [ADC_InjectedChannelConfig, 27](#)
 - [ADC_InjectedDiscModeCmd, 28](#)
 - [ADC_InjectedSequencerLengthConfig, 28](#)
 - [ADC_ITConfig, 29](#)
 - [ADC_MultiModeDMAResquestAfterLastTransferCmd, 29](#)
 - [ADC_RegularChannelConfig, 30](#)
 - [ADC_SetInjectedOffset, 31](#)
 - [ADC_SoftwareStartConv, 32](#)
 - [ADC_SoftwareStartInjectedConv, 32](#)
 - [ADC_StructInit, 32](#)
 - [ADC_TempSensorVrefintCmd, 33](#)
 - [ADC_VBATCmd, 33](#)
 - [CDR_ADDRESS, 12](#)
 - [CR1_AWDCH_RESET, 12](#)
 - [CR1_AWDMODE_RESET, 12](#)
 - [CR1_CLEAR_MASK, 12](#)
 - [CR1_DISCNUM_RESET, 12](#)
 - [CR2_CLEAR_MASK, 12](#)
 - [CR2_EXTEN_RESET, 12](#)
 - [CR2_JEXTEN_RESET, 12](#)
 - [CR2_JEXTSEL_RESET, 12](#)
 - [CR_CLEAR_MASK, 13](#)
 - [JDR_OFFSET, 13](#)
 - [JSQR_JL_RESET, 13](#)
 - [JSQR_JL_SET, 13](#)
 - [JSQR_JSQ_SET, 13](#)
 - [SMPR1_SMP_SET, 13](#)
 - [SMPR2_SMP_SET, 13](#)
 - [SQR1_L_RESET, 13](#)

- SQR1_SQ_SET, [13](#)
- SQR2_SQ_SET, [13](#)
- SQR3_SQ_SET, [14](#)
- adc
 - analogin_s, [316](#)
- adc.c
 - _ADC_ConfigChannel, [323](#)
 - _ADC_GetValue, [323](#)
 - _ADC_Init, [324](#)
 - _ADC_PollForConversion, [324](#)
 - _ADC_Start, [324](#)
 - _GPIO_Init, [324](#)
 - _adc_read, [324](#)
 - adc_init, [324](#)
 - adc_read, [324](#)
 - AdcHandle, [325](#)
 - analogin_init, [324](#)
 - pin_function, [324](#)
 - PinMap_ADC, [325](#)
 - pinmap_find_function, [324](#)
 - pinmap_find_peripheral, [325](#)
 - pinmap_function, [325](#)
 - pinmap_peripheral, [325](#)
 - pinmap_pinout, [325](#)
- adc.h
 - _ADC_ConfigChannel, [334](#)
 - _ADC_GET_FLAG, [328](#)
 - _ADC_GetValue, [334](#)
 - _ADC_Init, [334](#)
 - _ADC_PollForConversion, [334](#)
 - _ADC_Start, [335](#)
 - _GPIO_Init, [335](#)
 - _IS_BIT_CLR, [328](#)
 - _IS_BIT_SET, [328](#)
 - _adc_read, [335](#)
 - ADC1_BASE, [328](#)
 - ADC_1, [333](#)
 - ADC_CHANNEL_0, [328](#)
 - ADC_CHANNEL_1, [328](#)
 - ADC_CHANNEL_10, [328](#)
 - ADC_CHANNEL_11, [328](#)
 - ADC_CHANNEL_12, [328](#)
 - ADC_CHANNEL_13, [329](#)
 - ADC_CHANNEL_14, [329](#)
 - ADC_CHANNEL_15, [329](#)
 - ADC_CHANNEL_16, [329](#)
 - ADC_CHANNEL_17, [329](#)
 - ADC_CHANNEL_18, [329](#)
 - ADC_CHANNEL_2, [329](#)
 - ADC_CHANNEL_3, [329](#)
 - ADC_CHANNEL_4, [329](#)
 - ADC_CHANNEL_5, [329](#)
 - ADC_CHANNEL_6, [329](#)
 - ADC_CHANNEL_7, [329](#)
 - ADC_CHANNEL_8, [329](#)
 - ADC_CHANNEL_9, [329](#)
 - ADC_CHANNEL_TEMPSENSOR, [329](#)
 - ADC_CHANNEL_VBAT, [330](#)
 - ADC_CHANNEL_VREFINT, [330](#)
 - ADC_CR1_DISCONTINUOUS, [330](#)
 - ADC_CR1_SCANCONV, [330](#)
 - ADC_CR2_CONTINUOUS, [330](#)
 - ADC_CR2_DMAContReq, [330](#)
 - ADC_CR2_EOCSelection, [330](#)
 - adc_init, [335](#)
 - adc_read, [335](#)
 - ADC_SMPR1, [330](#)
 - ADC_SMPR2, [330](#)
 - ADC_SOFTWARE_START, [330](#)
 - ADC_SQR1, [330](#)
 - ADC_SQR1_RK, [330](#)
 - ADC_SQR2_RK, [331](#)
 - ADC_SQR3_RK, [331](#)
 - ADC_STAB_DELAY_US, [331](#)
 - ADCName, [333](#)
 - analogin_init, [335](#)
 - GPIO_MODE, [331](#)
 - GPIO_NOPULL, [331](#)
 - GPIO_PULLDOWN, [331](#)
 - GPIO_PULLUP, [331](#)
 - GPIO_SPEED_FAST, [331](#)
 - GPIO_SPEED_HIGH, [331](#)
 - GPIO_SPEED_LOW, [331](#)
 - GPIO_SPEED_MEDIUM, [331](#)
 - GPIOA_BASE, [332](#)
 - HAL_ADC_STATE_AWD, [334](#)
 - HAL_ADC_STATE_BUSY, [334](#)
 - HAL_ADC_STATE_BUSY_INJ, [334](#)
 - HAL_ADC_STATE_BUSY_INJ_REG, [334](#)
 - HAL_ADC_STATE_BUSY_REG, [334](#)
 - HAL_ADC_STATE_EOC, [334](#)
 - HAL_ADC_STATE_EOC_INJ, [334](#)
 - HAL_ADC_STATE_EOC_INJ_REG, [334](#)
 - HAL_ADC_STATE_EOC_REG, [334](#)
 - HAL_ADC_STATE_ERROR, [334](#)
 - HAL_ADC_STATE_READY, [333](#)
 - HAL_ADC_STATE_RESET, [333](#)
 - HAL_ADC_STATE_TIMEOUT, [334](#)
 - HAL_ADC_StateTypeDef, [333](#)
 - HAL_LOCKED, [334](#)
 - HAL_LockTypeDef, [334](#)
 - HAL_UNLOCKED, [334](#)
 - pin_function, [335](#)
 - pinmap_find_peripheral, [335](#)
 - pinmap_function, [335](#)
 - pinmap_peripheral, [335](#)
 - pinmap_pinout, [336](#)
 - PortA, [334](#)
 - PortB, [334](#)
 - PortC, [334](#)
 - PortD, [334](#)
 - PortE, [334](#)
 - PortH, [334](#)
 - PortName, [334](#)
 - RCC_ADC1_CLK_ENABLE, [332](#)
 - RCC_GPIOA_CLK_ENABLE, [332](#)

- RCC_GPIOB_CLK_ENABLE, [332](#)
- RCC_GPIOC_CLK_ENABLE, [332](#)
- STM_MODE_ANALOG, [332](#)
- STM_PIN, [332](#)
- STM_PIN_AFNUM, [332](#)
- STM_PIN_CHANNEL, [333](#)
- STM_PIN_DATA_EXT, [333](#)
- STM_PIN_MODE, [333](#)
- STM_PIN_PUPD, [333](#)
- STM_PORT, [333](#)
- UNUSED, [333](#)
- ADC1_BASE
 - adc.h, [328](#)
- ADC_1
 - adc.h, [333](#)
- ADC_analog_watchdog_selection, [82](#)
 - ADC_AnalogWatchdog_AllInjecEnable, [82](#)
 - ADC_AnalogWatchdog_AllRegAllInjecEnable, [82](#)
 - ADC_AnalogWatchdog_AllRegEnable, [82](#)
 - ADC_AnalogWatchdog_None, [82](#)
 - ADC_AnalogWatchdog_SingleInjecEnable, [82](#)
 - ADC_AnalogWatchdog_SingleRegEnable, [82](#)
 - ADC_AnalogWatchdog_SingleRegOrInjecEnable, [82](#)
 - IS_ADC_ANALOG_WATCHDOG, [82](#)
- ADC_AnalogWatchdog_AllInjecEnable
 - ADC_analog_watchdog_selection, [82](#)
- ADC_AnalogWatchdog_AllRegAllInjecEnable
 - ADC_analog_watchdog_selection, [82](#)
- ADC_AnalogWatchdog_AllRegEnable
 - ADC_analog_watchdog_selection, [82](#)
- ADC_AnalogWatchdog_None
 - ADC_analog_watchdog_selection, [82](#)
- ADC_AnalogWatchdog_SingleInjecEnable
 - ADC_analog_watchdog_selection, [82](#)
- ADC_AnalogWatchdog_SingleRegEnable
 - ADC_analog_watchdog_selection, [82](#)
- ADC_AnalogWatchdog_SingleRegOrInjecEnable
 - ADC_analog_watchdog_selection, [82](#)
- ADC_AnalogWatchdogCmd
 - ADC, [14](#)
 - Analog Watchdog configuration functions, [38](#)
- ADC_AnalogWatchdogSingleChannelConfig
 - ADC, [14](#)
 - Analog Watchdog configuration functions, [38](#)
- ADC_AnalogWatchdogThresholdsConfig
 - ADC, [15](#)
 - Analog Watchdog configuration functions, [39](#)
- ADC_AutoInjectedConvCmd
 - ADC, [16](#)
 - Injected channels Configuration functions, [50](#)
- ADC_BITS
 - platform.h, [352](#)
- ADC_CHANNEL_0
 - adc.h, [328](#)
- ADC_Channel_0
 - ADC_channels, [73](#)
- ADC_CHANNEL_1
 - adc.h, [328](#)
- ADC_Channel_1
 - ADC_channels, [73](#)
- ADC_CHANNEL_10
 - adc.h, [328](#)
- ADC_Channel_10
 - ADC_channels, [74](#)
- ADC_CHANNEL_11
 - adc.h, [328](#)
- ADC_Channel_11
 - ADC_channels, [74](#)
- ADC_CHANNEL_12
 - adc.h, [328](#)
- ADC_Channel_12
 - ADC_channels, [74](#)
- ADC_CHANNEL_13
 - adc.h, [329](#)
- ADC_Channel_13
 - ADC_channels, [74](#)
- ADC_CHANNEL_14
 - adc.h, [329](#)
- ADC_Channel_14
 - ADC_channels, [74](#)
- ADC_CHANNEL_15
 - adc.h, [329](#)
- ADC_Channel_15
 - ADC_channels, [74](#)
- ADC_CHANNEL_16
 - adc.h, [329](#)
- ADC_Channel_16
 - ADC_channels, [74](#)
- ADC_CHANNEL_17
 - adc.h, [329](#)
- ADC_Channel_17
 - ADC_channels, [74](#)
- ADC_CHANNEL_18
 - adc.h, [329](#)
- ADC_Channel_18
 - ADC_channels, [74](#)
- ADC_CHANNEL_2
 - adc.h, [329](#)
- ADC_Channel_2
 - ADC_channels, [74](#)
- ADC_CHANNEL_3
 - adc.h, [329](#)
- ADC_Channel_3
 - ADC_channels, [75](#)
- ADC_CHANNEL_4
 - adc.h, [329](#)
- ADC_Channel_4
 - ADC_channels, [75](#)
- ADC_CHANNEL_5
 - adc.h, [329](#)
- ADC_Channel_5
 - ADC_channels, [75](#)
- ADC_CHANNEL_6
 - adc.h, [329](#)
- ADC_Channel_6

- ADC_channels, 75
- ADC_CHANNEL_7
 - adc.h, 329
- ADC_Channel_7
 - ADC_channels, 75
- ADC_CHANNEL_8
 - adc.h, 329
- ADC_Channel_8
 - ADC_channels, 75
- ADC_CHANNEL_9
 - adc.h, 329
- ADC_Channel_9
 - ADC_channels, 75
- ADC_CHANNEL_TEMPSENSOR
 - adc.h, 329
- ADC_Channel_TempSensor
 - ADC_channels, 75
- ADC_CHANNEL_VBAT
 - adc.h, 330
- ADC_Channel_Vbat
 - ADC_channels, 75
- ADC_CHANNEL_VREFINT
 - adc.h, 330
- ADC_Channel_Vrefint
 - ADC_channels, 75
- ADC_ChannelConfTypeDef, 312
 - Channel, 313
 - Offset, 313
 - Rank, 313
 - SamplingTime, 313
- ADC_channels, 73
 - ADC_Channel_0, 73
 - ADC_Channel_1, 73
 - ADC_Channel_10, 74
 - ADC_Channel_11, 74
 - ADC_Channel_12, 74
 - ADC_Channel_13, 74
 - ADC_Channel_14, 74
 - ADC_Channel_15, 74
 - ADC_Channel_16, 74
 - ADC_Channel_17, 74
 - ADC_Channel_18, 74
 - ADC_Channel_2, 74
 - ADC_Channel_3, 75
 - ADC_Channel_4, 75
 - ADC_Channel_5, 75
 - ADC_Channel_6, 75
 - ADC_Channel_7, 75
 - ADC_Channel_8, 75
 - ADC_Channel_9, 75
 - ADC_Channel_TempSensor, 75
 - ADC_Channel_Vbat, 75
 - ADC_Channel_Vrefint, 75
 - IS_ADC_CHANNEL, 76
- ADC_ClearFlag
 - ADC, 16
 - Interrupts and flags management functions, 58
- ADC_ClearITPendingBit
 - ADC, 17
 - Interrupts and flags management functions, 58
- ADC_Cmd
 - ADC, 17
 - Initialization and Configuration functions, 35
- ADC_Common_mode, 62
 - ADC_DualMode_AlterTrig, 62
 - ADC_DualMode_InjecSimult, 62
 - ADC_DualMode_Interl, 62
 - ADC_DualMode_RegSimult, 62
 - ADC_DualMode_RegSimult_AlterTrig, 62
 - ADC_DualMode_RegSimult_InjecSimult, 62
 - ADC_Mode_Independent, 62
 - ADC_TripleMode_AlterTrig, 63
 - ADC_TripleMode_InjecSimult, 63
 - ADC_TripleMode_Interl, 63
 - ADC_TripleMode_RegSimult, 63
 - ADC_TripleMode_RegSimult_AlterTrig, 63
 - ADC_TripleMode_RegSimult_InjecSimult, 63
 - IS_ADC_MODE, 63
- ADC_CommonInit
 - ADC, 17
 - Initialization and Configuration functions, 35
- ADC_CommonInitTypeDef, 313
 - ADC_DMAAccessMode, 313
 - ADC_Mode, 313
 - ADC_Prescaler, 314
 - ADC_TwoSamplingDelay, 314
- ADC_CommonStructInit
 - ADC, 18
 - Initialization and Configuration functions, 35
- ADC_ContinuousConvMode
 - ADC_InitTypeDef, 315
- ADC_ContinuousModeCmd
 - ADC, 18
 - Regular Channels Configuration functions, 42
- ADC_CR1_DISCONTINUOUS
 - adc.h, 330
- ADC_CR1_SCANCONV
 - adc.h, 330
- ADC_CR2_CONTINUOUS
 - adc.h, 330
- ADC_CR2_DMAContReq
 - adc.h, 330
- ADC_CR2_EOCSelection
 - adc.h, 330
- ADC_data_align, 72
 - ADC_DataAlign_Left, 72
 - ADC_DataAlign_Right, 72
 - IS_ADC_DATA_ALIGN, 73
- ADC_DataAlign
 - ADC_InitTypeDef, 315
- ADC_DataAlign_Left
 - ADC_data_align, 72
- ADC_DataAlign_Right
 - ADC_data_align, 72
- ADC_DeInit
 - ADC, 19

- Initialization and Configuration functions, [36](#)
- ADC_delay_between_2_sampling_phases, [66](#)
 - ADC_TwoSamplingDelay_10Cycles, [66](#)
 - ADC_TwoSamplingDelay_11Cycles, [66](#)
 - ADC_TwoSamplingDelay_12Cycles, [66](#)
 - ADC_TwoSamplingDelay_13Cycles, [66](#)
 - ADC_TwoSamplingDelay_14Cycles, [66](#)
 - ADC_TwoSamplingDelay_15Cycles, [66](#)
 - ADC_TwoSamplingDelay_16Cycles, [67](#)
 - ADC_TwoSamplingDelay_17Cycles, [67](#)
 - ADC_TwoSamplingDelay_18Cycles, [67](#)
 - ADC_TwoSamplingDelay_19Cycles, [67](#)
 - ADC_TwoSamplingDelay_20Cycles, [67](#)
 - ADC_TwoSamplingDelay_5Cycles, [67](#)
 - ADC_TwoSamplingDelay_6Cycles, [67](#)
 - ADC_TwoSamplingDelay_7Cycles, [67](#)
 - ADC_TwoSamplingDelay_8Cycles, [67](#)
 - ADC_TwoSamplingDelay_9Cycles, [67](#)
 - IS_ADC_SAMPLING_DELAY, [68](#)
- ADC_Direct_memory_access_mode_for_multi_mode, [65](#)
 - ADC_DMAAccessMode_1, [65](#)
 - ADC_DMAAccessMode_2, [65](#)
 - ADC_DMAAccessMode_3, [65](#)
 - ADC_DMAAccessMode_Disabled, [65](#)
 - IS_ADC_DMA_ACCESS_MODE, [65](#)
- ADC_DiscModeChannelCountConfig
 - ADC, [19](#)
 - Regular Channels Configuration functions, [42](#)
- ADC_DiscModeCmd
 - ADC, [19](#)
 - Regular Channels Configuration functions, [43](#)
- ADC_DMAAccessMode
 - ADC_CommonInitTypeDef, [313](#)
- ADC_DMAAccessMode_1
 - ADC_Direct_memory_access_mode_for_multi_mode, [65](#)
- ADC_DMAAccessMode_2
 - ADC_Direct_memory_access_mode_for_multi_mode, [65](#)
- ADC_DMAAccessMode_3
 - ADC_Direct_memory_access_mode_for_multi_mode, [65](#)
- ADC_DMAAccessMode_Disabled
 - ADC_Direct_memory_access_mode_for_multi_mode, [65](#)
- ADC_DMAMCmd
 - ADC, [20](#)
 - Regular Channels DMA Configuration functions, [48](#)
- ADC_DMARequestAfterLastTransferCmd
 - ADC, [20](#)
 - Regular Channels DMA Configuration functions, [48](#)
- ADC_DualMode_AlterTrig
 - ADC_Common_mode, [62](#)
- ADC_DualMode_InjecSimult
 - ADC_Common_mode, [62](#)
- ADC_DualMode_Interl
 - ADC_Common_mode, [62](#)
- ADC_DualMode_RegSimult
 - ADC_Common_mode, [62](#)
- ADC_DualMode_RegSimult_AlterTrig
 - ADC_Common_mode, [62](#)
- ADC_DualMode_RegSimult_InjecSimult
 - ADC_Common_mode, [62](#)
- ADC_EOCOnEachRegularChannelCmd
 - ADC, [20](#)
 - Regular Channels Configuration functions, [43](#)
- ADC_Exported_Constants, [61](#)
 - IS_ADC_ALL_PERIPH, [61](#)
- ADC_external_trigger_edge_for_injected_channels_conversion, [77](#)
 - ADC_ExternalTrigInjecConvEdge_Falling, [78](#)
 - ADC_ExternalTrigInjecConvEdge_None, [78](#)
 - ADC_ExternalTrigInjecConvEdge_Rising, [78](#)
 - ADC_ExternalTrigInjecConvEdge_RisingFalling, [78](#)
 - IS_ADC_EXT_INJEC_TRIG_EDGE, [78](#)
- ADC_external_trigger_edge_for_regular_channels_conversion, [69](#)
 - ADC_ExternalTrigConvEdge_Falling, [69](#)
 - ADC_ExternalTrigConvEdge_None, [69](#)
 - ADC_ExternalTrigConvEdge_Rising, [69](#)
 - ADC_ExternalTrigConvEdge_RisingFalling, [69](#)
 - IS_ADC_EXT_TRIG_EDGE, [69](#)
- ADC_ExternalTrigConv
 - ADC_InitTypeDef, [315](#)
- ADC_ExternalTrigConv_Ext_IT11
 - ADC_extrenal_trigger_sources_for_regular_channels_conversion, [70](#)
- ADC_ExternalTrigConv_T1_CC1
 - ADC_extrenal_trigger_sources_for_regular_channels_conversion, [70](#)
- ADC_ExternalTrigConv_T1_CC2
 - ADC_extrenal_trigger_sources_for_regular_channels_conversion, [70](#)
- ADC_ExternalTrigConv_T1_CC3
 - ADC_extrenal_trigger_sources_for_regular_channels_conversion, [70](#)
- ADC_ExternalTrigConv_T2_CC2
 - ADC_extrenal_trigger_sources_for_regular_channels_conversion, [71](#)
- ADC_ExternalTrigConv_T2_CC3
 - ADC_extrenal_trigger_sources_for_regular_channels_conversion, [71](#)
- ADC_ExternalTrigConv_T2_CC4
 - ADC_extrenal_trigger_sources_for_regular_channels_conversion, [71](#)
- ADC_ExternalTrigConv_T2_TRGO
 - ADC_extrenal_trigger_sources_for_regular_channels_conversion, [71](#)
- ADC_ExternalTrigConv_T3_CC1
 - ADC_extrenal_trigger_sources_for_regular_channels_conversion, [71](#)
- ADC_ExternalTrigConv_T3_TRGO
 - ADC_extrenal_trigger_sources_for_regular_channels_conversion, [71](#)

- IS_ADC_EXT_INJEC_TRIG, [80](#)
- ADC_extenal_trigger_sources_for_regular_channels_conversion, [70](#)
 - ADC_ExternalTrigConv_Ext_IT11, [70](#)
 - ADC_ExternalTrigConv_T1_CC1, [70](#)
 - ADC_ExternalTrigConv_T1_CC2, [70](#)
 - ADC_ExternalTrigConv_T1_CC3, [70](#)
 - ADC_ExternalTrigConv_T2_CC2, [71](#)
 - ADC_ExternalTrigConv_T2_CC3, [71](#)
 - ADC_ExternalTrigConv_T2_CC4, [71](#)
 - ADC_ExternalTrigConv_T2_TRGO, [71](#)
 - ADC_ExternalTrigConv_T3_CC1, [71](#)
 - ADC_ExternalTrigConv_T3_TRGO, [71](#)
 - ADC_ExternalTrigConv_T4_CC4, [71](#)
 - ADC_ExternalTrigConv_T5_CC1, [71](#)
 - ADC_ExternalTrigConv_T5_CC2, [71](#)
 - ADC_ExternalTrigConv_T5_CC3, [71](#)
 - ADC_ExternalTrigConv_T8_CC1, [72](#)
 - ADC_ExternalTrigConv_T8_TRGO, [72](#)
 - IS_ADC_EXT_TRIG, [72](#)
- ADC_FLAG_AWD
 - ADC_flags_definition, [84](#)
- ADC_FLAG_EOC
 - ADC_flags_definition, [84](#)
- ADC_FLAG_JEOC
 - ADC_flags_definition, [84](#)
- ADC_FLAG_JSTRT
 - ADC_flags_definition, [84](#)
- ADC_FLAG_OVR
 - ADC_flags_definition, [84](#)
- ADC_FLAG_STRT
 - ADC_flags_definition, [84](#)
- ADC_flags_definition, [84](#)
 - ADC_FLAG_AWD, [84](#)
 - ADC_FLAG_EOC, [84](#)
 - ADC_FLAG_JEOC, [84](#)
 - ADC_FLAG_JSTRT, [84](#)
 - ADC_FLAG_OVR, [84](#)
 - ADC_FLAG_STRT, [84](#)
 - IS_ADC_CLEAR_FLAG, [84](#)
 - IS_ADC_GET_FLAG, [84](#)
- ADC_GetConversionValue
 - ADC, [23](#)
 - Regular Channels Configuration functions, [44](#)
- ADC_GetFlagStatus
 - ADC, [23](#)
 - Interrupts and flags management functions, [59](#)
- ADC_GetInjectedConversionValue
 - ADC, [24](#)
 - Injected channels Configuration functions, [52](#)
- ADC_GetITStatus
 - ADC, [24](#)
 - Interrupts and flags management functions, [59](#)
- ADC_GetMultiModeConversionValue
 - ADC, [25](#)
 - Regular Channels Configuration functions, [44](#)
- ADC_GetSoftwareStartConvStatus
 - ADC, [25](#)
 - Regular Channels Configuration functions, [44](#)
- ADC_GetSoftwareStartInjectedConvCmdStatus
 - ADC, [26](#)
 - Injected channels Configuration functions, [52](#)
- ADC_HandleTypeDef, [314](#)
 - ErrorCode, [314](#)
 - Init, [314](#)
 - Instance, [314](#)
 - Lock, [314](#)
 - NbrOfCurrentConversionRank, [314](#)
 - State, [314](#)
- ADC_Init
 - ADC, [26](#)
 - Initialization and Configuration functions, [36](#)
- adc_init
 - adc.c, [324](#)
 - adc.h, [335](#)
- ADC_InitTypeDef, [315](#)
 - ADC_ContinuousConvMode, [315](#)
 - ADC_DataAlign, [315](#)
 - ADC_ExternalTrigConv, [315](#)
 - ADC_ExternalTrigConvEdge, [315](#)
 - ADC_NbrOfConversion, [315](#)
 - ADC_Resolution, [315](#)
 - ADC_ScanConvMode, [316](#)
- ADC_injected_channel_selection, [81](#)
 - ADC_InjectedChannel_1, [81](#)
 - ADC_InjectedChannel_2, [81](#)
 - ADC_InjectedChannel_3, [81](#)
 - ADC_InjectedChannel_4, [81](#)
 - IS_ADC_INJECTED_CHANNEL, [81](#)
- ADC_injected_length, [85](#)
 - IS_ADC_INJECTED_LENGTH, [86](#)
- ADC_injected_offset, [85](#)
 - IS_ADC_OFFSET, [85](#)
- ADC_injected_rank, [86](#)
 - IS_ADC_INJECTED_RANK, [86](#)
- ADC_InjectedChannel_1
 - ADC_injected_channel_selection, [81](#)
- ADC_InjectedChannel_2
 - ADC_injected_channel_selection, [81](#)
- ADC_InjectedChannel_3
 - ADC_injected_channel_selection, [81](#)
- ADC_InjectedChannel_4
 - ADC_injected_channel_selection, [81](#)
- ADC_InjectedChannelConfig
 - ADC, [27](#)
 - Injected channels Configuration functions, [53](#)
- ADC_InjectedDiscModeCmd
 - ADC, [28](#)
 - Injected channels Configuration functions, [55](#)
- ADC_InjectedSequencerLengthConfig
 - ADC, [28](#)
 - Injected channels Configuration functions, [55](#)
- ADC_interrupts_definition, [83](#)
 - ADC_IT_AWD, [83](#)
 - ADC_IT_EOC, [83](#)
 - ADC_IT_JEOC, [83](#)

- ADC_IT_OVR, [83](#)
- IS_ADC_IT, [83](#)
- ADC_IT_AWD
 - ADC_interrupts_definition, [83](#)
- ADC_IT_EOC
 - ADC_interrupts_definition, [83](#)
- ADC_IT_JEOC
 - ADC_interrupts_definition, [83](#)
- ADC_IT_OVR
 - ADC_interrupts_definition, [83](#)
- ADC_ITConfig
 - ADC, [29](#)
 - Interrupts and flags management functions, [60](#)
- ADC_MASK
 - platform.h, [352](#)
- ADC_Mode
 - ADC_CommonInitTypeDef, [313](#)
- ADC_Mode_Independent
 - ADC_Common_mode, [62](#)
- ADC_MultiModeDMARequestAfterLastTransferCmd
 - ADC, [29](#)
 - Regular Channels DMA Configuration functions, [48](#)
- ADC_NbrOfConversion
 - ADC_InitTypeDef, [315](#)
- ADC_Prescaler, [64](#)
 - ADC_CommonInitTypeDef, [314](#)
 - ADC_Prescaler_Div2, [64](#)
 - ADC_Prescaler_Div4, [64](#)
 - ADC_Prescaler_Div6, [64](#)
 - ADC_Prescaler_Div8, [64](#)
 - IS_ADC_PRESCALER, [64](#)
- ADC_Prescaler_Div2
 - ADC_Prescaler, [64](#)
- ADC_Prescaler_Div4
 - ADC_Prescaler, [64](#)
- ADC_Prescaler_Div6
 - ADC_Prescaler, [64](#)
- ADC_Prescaler_Div8
 - ADC_Prescaler, [64](#)
- ADC_Private_Functions, [33](#)
- adc_read
 - adc.c, [324](#)
 - adc.h, [335](#)
- ADC_regular_discontinuous_mode_number, [87](#)
 - IS_ADC_REGULAR_DISC_NUMBER, [87](#)
- ADC_regular_length, [86](#)
 - IS_ADC_REGULAR_LENGTH, [86](#)
- ADC_regular_rank, [86](#)
 - IS_ADC_REGULAR_RANK, [87](#)
- ADC_RegularChannelConfig
 - ADC, [30](#)
 - Regular Channels Configuration functions, [45](#)
- ADC_Resolution
 - ADC_InitTypeDef, [315](#)
- ADC_resolution, [68](#)
 - ADC_Resolution_10b, [68](#)
 - ADC_Resolution_12b, [68](#)
 - ADC_Resolution_6b, [68](#)
 - ADC_Resolution_8b, [68](#)
 - IS_ADC_RESOLUTION, [69](#)
- ADC_Resolution_10b
 - ADC_resolution, [68](#)
- ADC_Resolution_12b
 - ADC_resolution, [68](#)
- ADC_Resolution_6b
 - ADC_resolution, [68](#)
- ADC_Resolution_8b
 - ADC_resolution, [68](#)
- ADC_SampleTime_112Cycles
 - ADC_sampling_times, [76](#)
- ADC_SampleTime_144Cycles
 - ADC_sampling_times, [76](#)
- ADC_SampleTime_15Cycles
 - ADC_sampling_times, [76](#)
- ADC_SampleTime_28Cycles
 - ADC_sampling_times, [77](#)
- ADC_SampleTime_3Cycles
 - ADC_sampling_times, [77](#)
- ADC_SampleTime_480Cycles
 - ADC_sampling_times, [77](#)
- ADC_SampleTime_56Cycles
 - ADC_sampling_times, [77](#)
- ADC_SampleTime_84Cycles
 - ADC_sampling_times, [77](#)
- ADC_sampling_times, [76](#)
 - ADC_SampleTime_112Cycles, [76](#)
 - ADC_SampleTime_144Cycles, [76](#)
 - ADC_SampleTime_15Cycles, [76](#)
 - ADC_SampleTime_28Cycles, [77](#)
 - ADC_SampleTime_3Cycles, [77](#)
 - ADC_SampleTime_480Cycles, [77](#)
 - ADC_SampleTime_56Cycles, [77](#)
 - ADC_SampleTime_84Cycles, [77](#)
 - IS_ADC_SAMPLE_TIME, [77](#)
- ADC_ScanConvMode
 - ADC_InitTypeDef, [316](#)
- ADC_SetInjectedOffset
 - ADC, [31](#)
 - Injected channels Configuration functions, [55](#)
- ADC_SMPR1
 - adc.h, [330](#)
- ADC_SMPR2
 - adc.h, [330](#)
- ADC_SOFTWARE_START
 - adc.h, [330](#)
- ADC_SoftwareStartConv
 - ADC, [32](#)
 - Regular Channels Configuration functions, [47](#)
- ADC_SoftwareStartInjectedConv
 - ADC, [32](#)
 - Injected channels Configuration functions, [56](#)
- ADC_SQR1
 - adc.h, [330](#)
- ADC_SQR1_RK
 - adc.h, [330](#)
- ADC_SQR2_RK

- adc.h, [331](#)
- ADC_SQR3_RK
 - adc.h, [331](#)
- ADC_STAB_DELAY_US
 - adc.h, [331](#)
- ADC_StructInit
 - ADC, [32](#)
 - Initialization and Configuration functions, [36](#)
- ADC_TempSensorVrefintCmd
 - ADC, [33](#)
 - Temperature Sensor, Vrefint (Voltage Reference internal), [40](#)
- ADC_thresholds, [85](#)
 - IS_ADC_THRESHOLD, [85](#)
- ADC_TripleMode_AlterTrig
 - ADC_Common_mode, [63](#)
- ADC_TripleMode_InjecSimult
 - ADC_Common_mode, [63](#)
- ADC_TripleMode_Interl
 - ADC_Common_mode, [63](#)
- ADC_TripleMode_RegSimult
 - ADC_Common_mode, [63](#)
- ADC_TripleMode_RegSimult_AlterTrig
 - ADC_Common_mode, [63](#)
- ADC_TripleMode_RegSimult_InjecSimult
 - ADC_Common_mode, [63](#)
- ADC_TwoSamplingDelay
 - ADC_CommonInitTypeDef, [314](#)
- ADC_TwoSamplingDelay_10Cycles
 - ADC_delay_between_2_sampling_phases, [66](#)
- ADC_TwoSamplingDelay_11Cycles
 - ADC_delay_between_2_sampling_phases, [66](#)
- ADC_TwoSamplingDelay_12Cycles
 - ADC_delay_between_2_sampling_phases, [66](#)
- ADC_TwoSamplingDelay_13Cycles
 - ADC_delay_between_2_sampling_phases, [66](#)
- ADC_TwoSamplingDelay_14Cycles
 - ADC_delay_between_2_sampling_phases, [66](#)
- ADC_TwoSamplingDelay_15Cycles
 - ADC_delay_between_2_sampling_phases, [66](#)
- ADC_TwoSamplingDelay_16Cycles
 - ADC_delay_between_2_sampling_phases, [67](#)
- ADC_TwoSamplingDelay_17Cycles
 - ADC_delay_between_2_sampling_phases, [67](#)
- ADC_TwoSamplingDelay_18Cycles
 - ADC_delay_between_2_sampling_phases, [67](#)
- ADC_TwoSamplingDelay_19Cycles
 - ADC_delay_between_2_sampling_phases, [67](#)
- ADC_TwoSamplingDelay_20Cycles
 - ADC_delay_between_2_sampling_phases, [67](#)
- ADC_TwoSamplingDelay_5Cycles
 - ADC_delay_between_2_sampling_phases, [67](#)
- ADC_TwoSamplingDelay_6Cycles
 - ADC_delay_between_2_sampling_phases, [67](#)
- ADC_TwoSamplingDelay_7Cycles
 - ADC_delay_between_2_sampling_phases, [67](#)
- ADC_TwoSamplingDelay_8Cycles
 - ADC_delay_between_2_sampling_phases, [67](#)
- ADC_TwoSamplingDelay_9Cycles
 - ADC_delay_between_2_sampling_phases, [67](#)
- ADC_VBATCmd
 - ADC, [33](#)
 - Temperature Sensor, Vrefint (Voltage Reference internal), [41](#)
- AdcHandle
 - adc.c, [325](#)
- ADCName
 - adc.h, [333](#)
- AHBPrescTable
 - STM32F4xx_System_Private_Variables, [308](#)
- Alternate
 - GPIO_InitTypeDef, [317](#)
- Analog Watchdog configuration functions, [37](#)
 - ADC_AnalogWatchdogCmd, [38](#)
 - ADC_AnalogWatchdogSingleChannelConfig, [38](#)
 - ADC_AnalogWatchdogThresholdsConfig, [39](#)
- analogin_init
 - adc.c, [324](#)
 - adc.h, [335](#)
- analogin_s, [316](#)
 - adc, [316](#)
 - channel, [316](#)
 - pin, [316](#)
- APBPrescTable
 - STM32F4xx_System_Private_Variables, [308](#)
- BDCR_ADDRESS
 - RCC, [182](#)
- BDCR_BDRST_BB
 - RCC, [182](#)
- BDCR_OFFSET
 - RCC, [182](#)
- BDCR_RTCEN_BB
 - RCC, [182](#)
- BDRST_BitNumber
 - RCC, [182](#)
- Bit_RESET
 - GPIO, [90](#)
- Bit_SET
 - GPIO, [90](#)
- BitAction
 - GPIO, [89](#)
- CDR_ADDRESS
 - ADC, [12](#)
- CFGR_I2SSRC_BB
 - RCC, [182](#)
- CFGR_MCO1_RESET_MASK
 - RCC, [183](#)
- CFGR_MCO2_RESET_MASK
 - RCC, [183](#)
- CFGR_OFFSET
 - RCC, [183](#)
- Channel
 - ADC_ChannelConfTypeDef, [313](#)
- channel
 - analogin_s, [316](#)

- CIR_BYTE2_ADDRESS
 - RCC, [183](#)
- CIR_BYTE3_ADDRESS
 - RCC, [183](#)
- CLK_FREQ
 - platform.h, [352](#)
- ClockPrescaler
 - _ADC_InitTypeDef, [311](#)
- CMSIS, [307](#)
- CMSIS_device_header
 - RTE_Components.h, [422](#)
- comparator.c
 - comparator_init, [339](#)
 - comparator_read, [339](#)
- comparator.h
 - comparator_init, [340](#)
 - comparator_read, [340](#)
 - comparator_set_callback, [340](#)
 - comparator_set_trigger, [340](#)
 - ComparatorTriggerMode, [339](#)
 - CompBoth, [339](#)
 - CompFalling, [339](#)
 - CompNone, [339](#)
 - CompRising, [339](#)
- comparator_init
 - comparator.c, [339](#)
 - comparator.h, [340](#)
- comparator_read
 - comparator.c, [339](#)
 - comparator.h, [340](#)
- comparator_set_callback
 - comparator.h, [340](#)
- comparator_set_trigger
 - comparator.h, [340](#)
- ComparatorTriggerMode
 - comparator.h, [339](#)
- CompBoth
 - comparator.h, [339](#)
- CompFalling
 - comparator.h, [339](#)
- CompNone
 - comparator.h, [339](#)
- CompRising
 - comparator.h, [339](#)
- ContinuousConvMode
 - _ADC_InitTypeDef, [311](#)
- CR1_AWDCH_RESET
 - ADC, [12](#)
- CR1_AWDMode_RESET
 - ADC, [12](#)
- CR1_CLEAR_MASK
 - ADC, [12](#)
 - I2C, [124](#)
 - USART, [266](#)
- CR1_DISCNUM_RESET
 - ADC, [12](#)
- CR2_CLEAR_MASK
 - ADC, [12](#)
- CR2_CLOCK_CLEAR_MASK
 - USART, [266](#)
- CR2_EXTEN_RESET
 - ADC, [12](#)
- CR2_JEXTEN_RESET
 - ADC, [12](#)
- CR2_JEXTSEL_RESET
 - ADC, [12](#)
- CR3_CLEAR_MASK
 - USART, [267](#)
- CR_BYTE3_ADDRESS
 - RCC, [183](#)
- CR_CLEAR_MASK
 - ADC, [13](#)
- CR_CSSON_BB
 - RCC, [183](#)
- CR_HSION_BB
 - RCC, [183](#)
- CR_OFFSET
 - RCC, [183](#)
- CR_PLLI2SON_BB
 - RCC, [183](#)
- CR_PLLON_BB
 - RCC, [183](#)
- CSR_LSION_BB
 - RCC, [183](#)
- CSR_OFFSET
 - RCC, [183](#)
- CSSON_BitNumber
 - RCC, [183](#)
- DAC_BITS
 - platform.h, [352](#)
- DAC_MASK
 - platform.h, [352](#)
- Data transfers functions, [149](#), [283](#)
 - I2C_ReceiveData, [149](#)
 - I2C_SendData, [149](#)
 - USART_ReceiveData, [283](#)
 - USART_SendData, [283](#)
- DataAlign
 - _ADC_InitTypeDef, [311](#)
- DiscontinuousConvMode
 - _ADC_InitTypeDef, [311](#)
- DMA transfers management functions, [152](#), [292](#)
 - I2C_DMAMCmd, [152](#)
 - I2C_DMALastTransferCmd, [153](#)
 - USART_DMAMCmd, [292](#)
- DMAContinuousRequests
 - _ADC_InitTypeDef, [311](#)
- drivers/adc.c, [323](#)
- drivers/adc.h, [325](#), [336](#)
- drivers/comparator.c, [338](#)
- drivers/comparator.h, [339](#), [340](#)
- drivers/gpio.c, [341](#)
- drivers/gpio.h, [345](#), [348](#)
- drivers/i2c.c, [349](#)
- drivers/i2c.h, [350](#), [351](#)
- drivers/platform.h, [351](#), [356](#)

- drivers/stm32f4xx_adc.c, 358
- drivers/stm32f4xx_adc.h, 361, 367
- drivers/stm32f4xx_gpio.c, 371
- drivers/stm32f4xx_gpio.h, 373, 377
- drivers/stm32f4xx_i2c.c, 380
- drivers/stm32f4xx_i2c.h, 383, 388
- drivers/stm32f4xx_rcc.c, 392
- drivers/stm32f4xx_rcc.h, 396, 402
- drivers/stm32f4xx_usart.c, 406
- drivers/stm32f4xx_usart.h, 409, 413
- drivers/timer.c, 415
- drivers/timer.h, 417, 418
- drivers/uart.c, 418
- drivers/uart.h, 420, 421
- EOCSelection
 - _ADC_InitTypeDef, 312
- ErrorCode
 - ADC_HandleTypeDef, 314
- ExternalTrigConv
 - _ADC_InitTypeDef, 312
- ExternalTrigConvEdge
 - _ADC_InitTypeDef, 312
- EXTI0_IRQHandler
 - gpio.c, 341
- EXTI15_10_IRQHandler
 - gpio.c, 341
- EXTI1_IRQHandler
 - gpio.c, 342
- EXTI2_IRQHandler
 - gpio.c, 342
- EXTI3_IRQHandler
 - gpio.c, 342
- EXTI4_IRQHandler
 - gpio.c, 342
- EXTI9_5_IRQHandler
 - gpio.c, 342
- EXTI_port_set
 - gpio.c, 344
- Falling
 - gpio.h, 346
- FLAG_MASK
 - I2C, 124
 - RCC, 183
- function
 - PinMap, 319
- GET_PIN_INDEX
 - platform.h, 352
- GET_PORT
 - platform.h, 352
- GET_PORT_INDEX
 - platform.h, 352
- GPIO, 87
 - Bit_RESET, 90
 - Bit_SET, 90
 - BitAction, 89
 - GPIO_DeInit, 92
 - GPIO_Init, 92
 - GPIO_Mode_AF, 90
 - GPIO_Mode_AN, 90
 - GPIO_Mode_IN, 90
 - GPIO_Mode_OUT, 90
 - GPIO_OType_OD, 90
 - GPIO_OType_PP, 90
 - GPIO_PinAFConfig, 92
 - GPIO_PinLockConfig, 95
 - GPIO_PuPd_DOWN, 90
 - GPIO_PuPd_NOPULL, 90
 - GPIO_PuPd_UP, 90
 - GPIO_ReadInputData, 95
 - GPIO_ReadInputDataBit, 95
 - GPIO_ReadOutputData, 96
 - GPIO_ReadOutputDataBit, 96
 - GPIO_ResetBits, 97
 - GPIO_SetBits, 97
 - GPIO_Speed_100MHz, 92
 - GPIO_Speed_25MHz, 92
 - GPIO_Speed_2MHz, 92
 - GPIO_Speed_50MHz, 92
 - GPIO_StructInit, 98
 - GPIO_ToggleBits, 98
 - GPIO_Write, 98
 - GPIO_WriteBit, 99
 - GPIO_Mode_TypeDef, 90
 - GPIOOType_TypeDef, 90
 - GPIOPuPd_TypeDef, 90
 - GPIOSpeed_TypeDef, 90
 - IS_GPIO_ALL_PERIPH, 89
 - IS_GPIO_BIT_ACTION, 89
 - IS_GPIO_MODE, 89
 - IS_GPIO_OTYPE, 89
 - IS_GPIO_PUPD, 89
 - IS_GPIO_SPEED, 89
- GPIO Alternate functions configuration function, 107
 - GPIO_PinAFConfig, 107
- GPIO Read and Write, 102
 - GPIO_ReadInputData, 102
 - GPIO_ReadInputDataBit, 103
 - GPIO_ReadOutputData, 103
 - GPIO_ReadOutputDataBit, 103
 - GPIO_ResetBits, 105
 - GPIO_SetBits, 105
 - GPIO_ToggleBits, 106
 - GPIO_Write, 106
 - GPIO_WriteBit, 106
- gpio.c
 - EXTI0_IRQHandler, 341
 - EXTI15_10_IRQHandler, 341
 - EXTI1_IRQHandler, 342
 - EXTI2_IRQHandler, 342
 - EXTI3_IRQHandler, 342
 - EXTI4_IRQHandler, 342
 - EXTI9_5_IRQHandler, 342
 - EXTI_port_set, 344
 - gpio_get, 342

- [gpio_get_range, 342](#)
 - [gpio_set, 343](#)
 - [gpio_set_callback, 343](#)
 - [gpio_set_mode, 343](#)
 - [gpio_set_range, 343](#)
 - [gpio_set_trigger, 344](#)
 - [gpio_toggle, 344](#)
 - [IRQ_pin_index, 344](#)
 - [IRQ_port_num, 344](#)
 - [IRQ_status, 344](#)
 - [priority, 344](#)
 - [prioritygroup, 344](#)
- gpio.h
 - [Falling, 346](#)
 - [gpio_get, 346](#)
 - [gpio_get_range, 346](#)
 - [gpio_set, 346](#)
 - [gpio_set_callback, 347](#)
 - [gpio_set_mode, 347](#)
 - [gpio_set_range, 347](#)
 - [gpio_set_trigger, 348](#)
 - [gpio_toggle, 348](#)
 - [Input, 345](#)
 - [None, 346](#)
 - [Output, 346](#)
 - [PinMode, 345](#)
 - [PullDown, 346](#)
 - [PullUp, 346](#)
 - [Reset, 345](#)
 - [Rising, 346](#)
 - [TriggerMode, 346](#)
- GPIO_AF_CAN1
 - [GPIO_Alternat_function_selection_define, 116](#)
- GPIO_AF_CAN2
 - [GPIO_Alternat_function_selection_define, 116](#)
- GPIO_AF_DCM1
 - [GPIO_Alternat_function_selection_define, 117](#)
- GPIO_AF_ETH
 - [GPIO_Alternat_function_selection_define, 117](#)
- GPIO_AF_EVENTOUT
 - [GPIO_Alternat_function_selection_define, 117](#)
- GPIO_AF_FSMC
 - [GPIO_Alternat_function_selection_define, 117](#)
- GPIO_AF_I2C1
 - [GPIO_Alternat_function_selection_define, 117](#)
- GPIO_AF_I2C2
 - [GPIO_Alternat_function_selection_define, 117](#)
- GPIO_AF_I2C3
 - [GPIO_Alternat_function_selection_define, 117](#)
- GPIO_AF_I2S3ext
 - [GPIO_Alternat_function_selection_define, 117](#)
- GPIO_AF_MCO
 - [GPIO_Alternat_function_selection_define, 118](#)
- GPIO_AF_OTG1_FS
 - [GPIO_Legacy, 122](#)
- GPIO_AF_OTG2_FS
 - [GPIO_Legacy, 122](#)
- GPIO_AF_OTG2_HS
 - [GPIO_Legacy, 122](#)
- GPIO_AF_OTG_FS
 - [GPIO_Alternat_function_selection_define, 118](#)
- GPIO_AF_OTG_HS
 - [GPIO_Alternat_function_selection_define, 118](#)
- GPIO_AF_OTG_HS_FS
 - [GPIO_Alternat_function_selection_define, 118](#)
- GPIO_AF_RTC_50Hz
 - [GPIO_Alternat_function_selection_define, 118](#)
- GPIO_AF_SDIO
 - [GPIO_Alternat_function_selection_define, 118](#)
- GPIO_AF_SPI1
 - [GPIO_Alternat_function_selection_define, 118](#)
- GPIO_AF_SPI2
 - [GPIO_Alternat_function_selection_define, 118](#)
- GPIO_AF_SPI3
 - [GPIO_Alternat_function_selection_define, 119](#)
- GPIO_AF_SWJ
 - [GPIO_Alternat_function_selection_define, 119](#)
- GPIO_AF_TAMPER
 - [GPIO_Alternat_function_selection_define, 119](#)
- GPIO_AF_TIM1
 - [GPIO_Alternat_function_selection_define, 119](#)
- GPIO_AF_TIM10
 - [GPIO_Alternat_function_selection_define, 119](#)
- GPIO_AF_TIM11
 - [GPIO_Alternat_function_selection_define, 119](#)
- GPIO_AF_TIM12
 - [GPIO_Alternat_function_selection_define, 119](#)
- GPIO_AF_TIM13
 - [GPIO_Alternat_function_selection_define, 119](#)
- GPIO_AF_TIM14
 - [GPIO_Alternat_function_selection_define, 119](#)
- GPIO_AF_TIM2
 - [GPIO_Alternat_function_selection_define, 120](#)
- GPIO_AF_TIM3
 - [GPIO_Alternat_function_selection_define, 120](#)
- GPIO_AF_TIM4
 - [GPIO_Alternat_function_selection_define, 120](#)
- GPIO_AF_TIM5
 - [GPIO_Alternat_function_selection_define, 120](#)
- GPIO_AF_TIM8
 - [GPIO_Alternat_function_selection_define, 120](#)
- GPIO_AF_TIM9
 - [GPIO_Alternat_function_selection_define, 120](#)
- GPIO_AF_TRACE
 - [GPIO_Alternat_function_selection_define, 120](#)
- GPIO_AF_UART4
 - [GPIO_Alternat_function_selection_define, 120](#)
- GPIO_AF_UART5
 - [GPIO_Alternat_function_selection_define, 120](#)
- GPIO_AF_USART1
 - [GPIO_Alternat_function_selection_define, 121](#)
- GPIO_AF_USART2
 - [GPIO_Alternat_function_selection_define, 121](#)
- GPIO_AF_USART3
 - [GPIO_Alternat_function_selection_define, 121](#)
- GPIO_AF_USART6
 - [GPIO_Alternat_function_selection_define, 121](#)

- GPIO_Alternat_function_selection_define, [121](#)
- GPIO_Alternat_function_selection_define, [115](#)
- GPIO_AF_CAN1, [116](#)
- GPIO_AF_CAN2, [116](#)
- GPIO_AF_DCMI, [117](#)
- GPIO_AF_ETH, [117](#)
- GPIO_AF_EVENTOUT, [117](#)
- GPIO_AF_FSMC, [117](#)
- GPIO_AF_I2C1, [117](#)
- GPIO_AF_I2C2, [117](#)
- GPIO_AF_I2C3, [117](#)
- GPIO_AF_I2S3ext, [117](#)
- GPIO_AF_MCO, [118](#)
- GPIO_AF_OTG_FS, [118](#)
- GPIO_AF_OTG_HS, [118](#)
- GPIO_AF_OTG_HS_FS, [118](#)
- GPIO_AF_RTC_50Hz, [118](#)
- GPIO_AF_SDIO, [118](#)
- GPIO_AF_SPI1, [118](#)
- GPIO_AF_SPI2, [118](#)
- GPIO_AF_SPI3, [119](#)
- GPIO_AF_SWJ, [119](#)
- GPIO_AF_TAMPER, [119](#)
- GPIO_AF_TIM1, [119](#)
- GPIO_AF_TIM10, [119](#)
- GPIO_AF_TIM11, [119](#)
- GPIO_AF_TIM12, [119](#)
- GPIO_AF_TIM13, [119](#)
- GPIO_AF_TIM14, [119](#)
- GPIO_AF_TIM2, [120](#)
- GPIO_AF_TIM3, [120](#)
- GPIO_AF_TIM4, [120](#)
- GPIO_AF_TIM5, [120](#)
- GPIO_AF_TIM8, [120](#)
- GPIO_AF_TIM9, [120](#)
- GPIO_AF_TRACE, [120](#)
- GPIO_AF_UART4, [120](#)
- GPIO_AF_UART5, [120](#)
- GPIO_AF_USART1, [121](#)
- GPIO_AF_USART2, [121](#)
- GPIO_AF_USART3, [121](#)
- GPIO_AF_USART6, [121](#)
- IS_GPIO_AF, [121](#)
- GPIO_DeInit
 - GPIO, [92](#)
 - Initialization and Configuration, [100](#)
- GPIO_Exported_Constants, [110](#)
- gpio_get
 - gpio.c, [342](#)
 - gpio.h, [346](#)
- gpio_get_range
 - gpio.c, [342](#)
 - gpio.h, [346](#)
- GPIO_Init
 - GPIO, [92](#)
 - Initialization and Configuration, [100](#)
- GPIO_InitTypeDef, [316](#)
 - Alternate, [317](#)
 - GPIO_Mode, [317](#)
 - GPIO_OType, [317](#)
 - GPIO_Pin, [317](#)
 - GPIO_PuPd, [317](#)
 - GPIO_Speed, [317](#)
 - Mode, [317](#)
 - Pin, [317](#)
 - Pull, [317](#)
 - Speed, [317](#)
- GPIO_Legacy, [121](#)
 - GPIO_AF_OTG1_FS, [122](#)
 - GPIO_AF_OTG2_FS, [122](#)
 - GPIO_AF_OTG2_HS, [122](#)
 - GPIO_Mode_AIN, [122](#)
- GPIO_MODE
 - adc.h, [331](#)
- GPIO_Mode
 - GPIO_InitTypeDef, [317](#)
- GPIO_Mode_AF
 - GPIO, [90](#)
- GPIO_Mode_AIN
 - GPIO_Legacy, [122](#)
- GPIO_Mode_AN
 - GPIO, [90](#)
- GPIO_Mode_IN
 - GPIO, [90](#)
- GPIO_Mode_OUT
 - GPIO, [90](#)
- GPIO_NOPULL
 - adc.h, [331](#)
- GPIO_OType
 - GPIO_InitTypeDef, [317](#)
- GPIO_OType_OD
 - GPIO, [90](#)
- GPIO_OType_PP
 - GPIO, [90](#)
- GPIO_Pin
 - GPIO_InitTypeDef, [317](#)
- GPIO_Pin_0
 - GPIO_pins_define, [110](#)
- GPIO_Pin_1
 - GPIO_pins_define, [110](#)
- GPIO_Pin_10
 - GPIO_pins_define, [111](#)
- GPIO_Pin_11
 - GPIO_pins_define, [111](#)
- GPIO_Pin_12
 - GPIO_pins_define, [111](#)
- GPIO_Pin_13
 - GPIO_pins_define, [111](#)
- GPIO_Pin_14
 - GPIO_pins_define, [111](#)
- GPIO_Pin_15
 - GPIO_pins_define, [111](#)
- GPIO_Pin_2
 - GPIO_pins_define, [111](#)
- GPIO_Pin_3
 - GPIO_pins_define, [111](#)

- GPIO_Pin_4
 - GPIO_pins_define, [111](#)
- GPIO_Pin_5
 - GPIO_pins_define, [111](#)
- GPIO_Pin_6
 - GPIO_pins_define, [112](#)
- GPIO_Pin_7
 - GPIO_pins_define, [112](#)
- GPIO_Pin_8
 - GPIO_pins_define, [112](#)
- GPIO_Pin_9
 - GPIO_pins_define, [112](#)
- GPIO_Pin_All
 - GPIO_pins_define, [112](#)
- GPIO_Pin_sources, [113](#)
 - GPIO_PinSource0, [113](#)
 - GPIO_PinSource1, [113](#)
 - GPIO_PinSource10, [113](#)
 - GPIO_PinSource11, [113](#)
 - GPIO_PinSource12, [113](#)
 - GPIO_PinSource13, [113](#)
 - GPIO_PinSource14, [114](#)
 - GPIO_PinSource15, [114](#)
 - GPIO_PinSource2, [114](#)
 - GPIO_PinSource3, [114](#)
 - GPIO_PinSource4, [114](#)
 - GPIO_PinSource5, [114](#)
 - GPIO_PinSource6, [114](#)
 - GPIO_PinSource7, [114](#)
 - GPIO_PinSource8, [114](#)
 - GPIO_PinSource9, [114](#)
 - IS_GPIO_PIN_SOURCE, [115](#)
- GPIO_PinAFConfig
 - GPIO, [92](#)
 - GPIO Alternate functions configuration function, [107](#)
- GPIO_PinLockConfig
 - GPIO, [95](#)
 - Initialization and Configuration, [101](#)
- GPIO_pins_define, [110](#)
 - GPIO_Pin_0, [110](#)
 - GPIO_Pin_1, [110](#)
 - GPIO_Pin_10, [111](#)
 - GPIO_Pin_11, [111](#)
 - GPIO_Pin_12, [111](#)
 - GPIO_Pin_13, [111](#)
 - GPIO_Pin_14, [111](#)
 - GPIO_Pin_15, [111](#)
 - GPIO_Pin_2, [111](#)
 - GPIO_Pin_3, [111](#)
 - GPIO_Pin_4, [111](#)
 - GPIO_Pin_5, [111](#)
 - GPIO_Pin_6, [112](#)
 - GPIO_Pin_7, [112](#)
 - GPIO_Pin_8, [112](#)
 - GPIO_Pin_9, [112](#)
 - GPIO_Pin_All, [112](#)
 - IS_GET_GPIO_PIN, [112](#)
 - IS_GPIO_PIN, [112](#)
- GPIO_PinSource0
 - GPIO_Pin_sources, [113](#)
- GPIO_PinSource1
 - GPIO_Pin_sources, [113](#)
- GPIO_PinSource10
 - GPIO_Pin_sources, [113](#)
- GPIO_PinSource11
 - GPIO_Pin_sources, [113](#)
- GPIO_PinSource12
 - GPIO_Pin_sources, [113](#)
- GPIO_PinSource13
 - GPIO_Pin_sources, [113](#)
- GPIO_PinSource14
 - GPIO_Pin_sources, [114](#)
- GPIO_PinSource15
 - GPIO_Pin_sources, [114](#)
- GPIO_PinSource2
 - GPIO_Pin_sources, [114](#)
- GPIO_PinSource3
 - GPIO_Pin_sources, [114](#)
- GPIO_PinSource4
 - GPIO_Pin_sources, [114](#)
- GPIO_PinSource5
 - GPIO_Pin_sources, [114](#)
- GPIO_PinSource6
 - GPIO_Pin_sources, [114](#)
- GPIO_PinSource7
 - GPIO_Pin_sources, [114](#)
- GPIO_PinSource8
 - GPIO_Pin_sources, [114](#)
- GPIO_PinSource9
 - GPIO_Pin_sources, [114](#)
- GPIO_Private_Functions, [99](#)
- GPIO_PULLDOWN
 - adc.h, [331](#)
- GPIO_PULLUP
 - adc.h, [331](#)
- GPIO_PuPd
 - GPIO_InitTypeDef, [317](#)
- GPIO_PuPd_DOWN
 - GPIO, [90](#)
- GPIO_PuPd_NOPULL
 - GPIO, [90](#)
- GPIO_PuPd_UP
 - GPIO, [90](#)
- GPIO_ReadInputData
 - GPIO, [95](#)
 - GPIO Read and Write, [102](#)
- GPIO_ReadInputDataBit
 - GPIO, [95](#)
 - GPIO Read and Write, [103](#)
- GPIO_ReadOutputData
 - GPIO, [96](#)
 - GPIO Read and Write, [103](#)
- GPIO_ReadOutputDataBit
 - GPIO, [96](#)
 - GPIO Read and Write, [103](#)

- GPIO_ResetBits
 - GPIO, [97](#)
 - GPIO Read and Write, [105](#)
- gpio_set
 - gpio.c, [343](#)
 - gpio.h, [346](#)
- gpio_set_callback
 - gpio.c, [343](#)
 - gpio.h, [347](#)
- gpio_set_mode
 - gpio.c, [343](#)
 - gpio.h, [347](#)
- gpio_set_range
 - gpio.c, [343](#)
 - gpio.h, [347](#)
- gpio_set_trigger
 - gpio.c, [344](#)
 - gpio.h, [348](#)
- GPIO_SetBits
 - GPIO, [97](#)
 - GPIO Read and Write, [105](#)
- GPIO_Speed
 - GPIO_InitTypeDef, [317](#)
- GPIO_Speed_100MHz
 - GPIO, [92](#)
- GPIO_Speed_25MHz
 - GPIO, [92](#)
- GPIO_Speed_2MHz
 - GPIO, [92](#)
- GPIO_Speed_50MHz
 - GPIO, [92](#)
- GPIO_SPEED_FAST
 - adc.h, [331](#)
- GPIO_SPEED_HIGH
 - adc.h, [331](#)
- GPIO_SPEED_LOW
 - adc.h, [331](#)
- GPIO_SPEED_MEDIUM
 - adc.h, [331](#)
- GPIO_StructInit
 - GPIO, [98](#)
 - Initialization and Configuration, [101](#)
- gpio_toggle
 - gpio.c, [344](#)
 - gpio.h, [348](#)
- GPIO_ToggleBits
 - GPIO, [98](#)
 - GPIO Read and Write, [106](#)
- GPIO_Write
 - GPIO, [98](#)
 - GPIO Read and Write, [106](#)
- GPIO_WriteBit
 - GPIO, [99](#)
 - GPIO Read and Write, [106](#)
- GPIOA_BASE
 - adc.h, [332](#)
- GPIO_Mode_TypeDef
 - GPIO, [90](#)
- GPIOType_TypeDef
 - GPIO, [90](#)
- GPIO_PuPd_TypeDef
 - GPIO, [90](#)
- GPIO_Speed_TypeDef
 - GPIO, [90](#)
- HAL_ADC_STATE_AWD
 - adc.h, [334](#)
- HAL_ADC_STATE_BUSY
 - adc.h, [334](#)
- HAL_ADC_STATE_BUSY_INJ
 - adc.h, [334](#)
- HAL_ADC_STATE_BUSY_INJ_REG
 - adc.h, [334](#)
- HAL_ADC_STATE_BUSY_REG
 - adc.h, [334](#)
- HAL_ADC_STATE_EOC
 - adc.h, [334](#)
- HAL_ADC_STATE_EOC_INJ
 - adc.h, [334](#)
- HAL_ADC_STATE_EOC_INJ_REG
 - adc.h, [334](#)
- HAL_ADC_STATE_EOC_REG
 - adc.h, [334](#)
- HAL_ADC_STATE_ERROR
 - adc.h, [334](#)
- HAL_ADC_STATE_READY
 - adc.h, [333](#)
- HAL_ADC_STATE_RESET
 - adc.h, [333](#)
- HAL_ADC_STATE_TIMEOUT
 - adc.h, [334](#)
- HAL_ADC_StateTypeDef
 - adc.h, [333](#)
- HAL_LOCKED
 - adc.h, [334](#)
- HAL_LockTypeDef
 - adc.h, [334](#)
- HAL_UNLOCKED
 - adc.h, [334](#)
- Halfduplex mode function, [287](#)
 - USART_HalfDuplexCmd, [288](#)
- HCLK_Frequency
 - RCC_ClocksTypeDef, [319](#)
- HSE_STARTUP_TIMEOUT
 - stm32f4xx_rcc.c, [396](#)
- HSE_VALUE
 - stm32f4xx_rcc.c, [396](#)
 - STM32F4xx_System_Private_Includes, [308](#)
- HSI_VALUE
 - stm32f4xx_rcc.c, [396](#)
 - STM32F4xx_System_Private_Includes, [308](#)
- HSION_BitNumber
 - RCC, [184](#)
- I2C, [122](#)
 - CR1_CLEAR_MASK, [124](#)
 - FLAG_MASK, [124](#)

- I2C_AcknowledgeConfig, 124
- I2C_ARPCmd, 125
- I2C_CalculatePEC, 125
- I2C_CheckEvent, 125
- I2C_ClearFlag, 127
- I2C_ClearITPendingBit, 127
- I2C_Cmd, 128
- I2C_DeInit, 129
- I2C_DMAMCmd, 129
- I2C_DMALastTransferCmd, 129
- I2C_DualAddressCmd, 130
- I2C_FastModeDutyCycleConfig, 130
- I2C_GeneralCallCmd, 130
- I2C_GenerateSTART, 131
- I2C_GenerateSTOP, 131
- I2C_GetFlagStatus, 131
- I2C_GetITStatus, 132
- I2C_GetLastEvent, 133
- I2C_GetPEC, 134
- I2C_Init, 134
- I2C_ITConfig, 134
- I2C_NACKPositionConfig, 135
- I2C_OwnAddress2Config, 136
- I2C_PECPositionConfig, 136
- I2C_ReadRegister, 137
- I2C_ReceiveData, 137
- I2C_Send7bitAddress, 138
- I2C_SendData, 138
- I2C_SMBusAlertConfig, 138
- I2C_SoftwareResetCmd, 139
- I2C_StretchClockCmd, 139
- I2C_StructInit, 140
- I2C_TransmitPEC, 140
- ITEN_MASK, 124
- i2c.c
 - i2c_init, 349
 - i2c_read, 349
 - i2c_write, 349
- i2c.h
 - i2c_init, 350
 - i2c_read, 350
 - i2c_write, 350
- I2C_Ack
 - I2C_InitTypeDef, 318
- I2C_Ack_Disable
 - I2C_acknowledgement, 164
- I2C_Ack_Enable
 - I2C_acknowledgement, 164
- I2C_AcknowledgeConfig
 - I2C, 124
 - Initialization and Configuration functions, 142
- I2C_acknowledged_address, 165
 - I2C_AcknowledgedAddress_10bit, 165
 - I2C_AcknowledgedAddress_7bit, 165
 - IS_I2C_ACKNOWLEDGE_ADDRESS, 166
- I2C_AcknowledgedAddress
 - I2C_InitTypeDef, 318
- I2C_AcknowledgedAddress_10bit
 - I2C_acknowledged_address, 165
- I2C_AcknowledgedAddress_7bit
 - I2C_acknowledged_address, 165
- I2C_acknowledgement, 164
 - I2C_Ack_Disable, 164
 - I2C_Ack_Enable, 164
 - IS_I2C_ACK_STATE, 164
- I2C_ARPCmd
 - I2C, 125
 - Initialization and Configuration functions, 142
- I2C_CalculatePEC
 - I2C, 125
 - PEC management functions, 150
- I2C_CheckEvent
 - I2C, 125
 - Interrupts events and flags management functions, 155
- I2C_ClearFlag
 - I2C, 127
 - Interrupts events and flags management functions, 156
- I2C_ClearITPendingBit
 - I2C, 127
 - Interrupts events and flags management functions, 157
- I2C_clock_speed, 179
 - IS_I2C_CLOCK_SPEED, 179
- I2C_ClockSpeed
 - I2C_InitTypeDef, 318
- I2C_Cmd
 - I2C, 128
 - Initialization and Configuration functions, 142
- I2C_DeInit
 - I2C, 129
 - Initialization and Configuration functions, 143
- I2C_Direction_Receiver
 - I2C_transfer_direction, 165
- I2C_Direction_Transmitter
 - I2C_transfer_direction, 165
- I2C_DMAMCmd
 - DMA transfers management functions, 152
 - I2C, 129
- I2C_DMALastTransferCmd
 - DMA transfers management functions, 153
 - I2C, 129
- I2C_DualAddressCmd
 - I2C, 130
 - Initialization and Configuration functions, 143
- I2C_duty_cycle_in_fast_mode, 163
 - I2C_DutyCycle_16_9, 163
 - I2C_DutyCycle_2, 163
 - IS_I2C_DUTY_CYCLE, 164
- I2C_DutyCycle
 - I2C_InitTypeDef, 318
- I2C_DutyCycle_16_9
 - I2C_duty_cycle_in_fast_mode, 163
- I2C_DutyCycle_2
 - I2C_duty_cycle_in_fast_mode, 163

- I2C_EVENT_MASTER_BYTE_RECEIVED
 - I2C_Events, 176
- I2C_EVENT_MASTER_BYTE_TRANSMITTED
 - I2C_Events, 176
- I2C_EVENT_MASTER_BYTE_TRANSMITTING
 - I2C_Events, 176
- I2C_EVENT_MASTER_MODE_ADDRESS10
 - I2C_Events, 176
- I2C_EVENT_MASTER_MODE_SELECT
 - I2C_Events, 177
- I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED
 - I2C_Events, 177
- I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED
 - I2C_Events, 177
- I2C_EVENT_SLAVE_ACK_FAILURE
 - I2C_Events, 177
- I2C_EVENT_SLAVE_BYTE_RECEIVED
 - I2C_Events, 177
- I2C_EVENT_SLAVE_BYTE_TRANSMITTED
 - I2C_Events, 178
- I2C_EVENT_SLAVE_BYTE_TRANSMITTING
 - I2C_Events, 178
- I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED
 - I2C_Events, 178
- I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED
 - I2C_Events, 178
- I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED
 - I2C_Events, 178
- I2C_EVENT_SLAVE_STOP_DETECTED
 - I2C_Events, 178
- I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED
 - I2C_Events, 178
- I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED
 - I2C_Events, 179
- I2C_Events, 175
 - I2C_EVENT_MASTER_BYTE_RECEIVED, 176
 - I2C_EVENT_MASTER_BYTE_TRANSMITTED, 176
 - I2C_EVENT_MASTER_BYTE_TRANSMITTING, 176
 - I2C_EVENT_MASTER_MODE_ADDRESS10, 176
 - I2C_EVENT_MASTER_MODE_SELECT, 177
 - I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED, 177
 - I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED, 177
 - I2C_EVENT_SLAVE_ACK_FAILURE, 177
 - I2C_EVENT_SLAVE_BYTE_RECEIVED, 177
 - I2C_EVENT_SLAVE_BYTE_TRANSMITTED, 178
 - I2C_EVENT_SLAVE_BYTE_TRANSMITTING, 178
 - I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED, 178
 - I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED, 178
 - I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED, 178
 - I2C_EVENT_SLAVE_STOP_DETECTED, 178
- I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED, 178
- I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED, 179
- IS_I2C_EVENT, 179
- I2C_Exported_Constants, 162
- IS_I2C_ALL_PERIPH, 162
- I2C_FastModeDutyCycleConfig
 - I2C, 130
- Initialization and Configuration functions, 143
- I2C_FLAG_ADD10
 - I2C_flags_definition, 172
- I2C_FLAG_ADDR
 - I2C_flags_definition, 172
- I2C_FLAG_AF
 - I2C_flags_definition, 173
- I2C_FLAG_ARLO
 - I2C_flags_definition, 173
- I2C_FLAG_BERR
 - I2C_flags_definition, 173
- I2C_FLAG_BTF
 - I2C_flags_definition, 173
- I2C_FLAG_BUSY
 - I2C_flags_definition, 173
- I2C_FLAG_DUALF
 - I2C_flags_definition, 173
- I2C_FLAG_GENCALL
 - I2C_flags_definition, 173
- I2C_FLAG_MSL
 - I2C_flags_definition, 173
- I2C_FLAG_OVR
 - I2C_flags_definition, 173
- I2C_FLAG_PECERR
 - I2C_flags_definition, 173
- I2C_FLAG_RXNE
 - I2C_flags_definition, 174
- I2C_FLAG_SB
 - I2C_flags_definition, 174
- I2C_FLAG_SMBALERT
 - I2C_flags_definition, 174
- I2C_FLAG_SMBDEFAULT
 - I2C_flags_definition, 174
- I2C_FLAG_SMBHOST
 - I2C_flags_definition, 174
- I2C_FLAG_STOPF
 - I2C_flags_definition, 174
- I2C_FLAG_TIMEOUT
 - I2C_flags_definition, 174
- I2C_FLAG_TRA
 - I2C_flags_definition, 174
- I2C_FLAG_TXE
 - I2C_flags_definition, 174
- I2C_FLAG_ADD10, 172
- I2C_FLAG_ADDR, 172
- I2C_FLAG_AF, 173
- I2C_FLAG_ARLO, 173
- I2C_FLAG_BERR, 173

- I2C_FLAG_BTFF, 173
- I2C_FLAG_BUSY, 173
- I2C_FLAG_DUALF, 173
- I2C_FLAG_GENCALL, 173
- I2C_FLAG_MSL, 173
- I2C_FLAG_OVR, 173
- I2C_FLAG_PECERR, 173
- I2C_FLAG_RXNE, 174
- I2C_FLAG_SB, 174
- I2C_FLAG_SMBALERT, 174
- I2C_FLAG_SMBDEFAULT, 174
- I2C_FLAG_SMBHOST, 174
- I2C_FLAG_STOPF, 174
- I2C_FLAG_TIMEOUT, 174
- I2C_FLAG_TRA, 174
- I2C_FLAG_TXE, 174
- IS_I2C_CLEAR_FLAG, 174
- IS_I2C_GET_FLAG, 175
- I2C_GeneralCallCmd
 - I2C, 130
 - Initialization and Configuration functions, 144
- I2C_GenerateSTART
 - I2C, 131
 - Initialization and Configuration functions, 144
- I2C_GenerateSTOP
 - I2C, 131
 - Initialization and Configuration functions, 145
- I2C_GetFlagStatus
 - I2C, 131
 - Interrupts events and flags management functions, 158
- I2C_GetITStatus
 - I2C, 132
 - Interrupts events and flags management functions, 159
- I2C_GetLastEvent
 - I2C, 133
 - Interrupts events and flags management functions, 160
- I2C_GetPEC
 - I2C, 134
 - PEC management functions, 151
- I2C_Init
 - I2C, 134
 - Initialization and Configuration functions, 145
- i2c_init
 - i2c.c, 349
 - i2c.h, 350
- I2C_InitTypeDef, 318
 - I2C_Ack, 318
 - I2C_AcknowledgedAddress, 318
 - I2C_ClockSpeed, 318
 - I2C_DutyCycle, 318
 - I2C_Mode, 318
 - I2C_OwnAddress1, 318
- I2C_interrupts_definition, 169
 - I2C_IT_ADD10, 170
 - I2C_IT_ADDR, 170
 - I2C_IT_AF, 170
 - I2C_IT_ARLO, 170
 - I2C_IT_BERR, 170
 - I2C_IT_BTFF, 170
 - I2C_IT_BUF, 170
 - I2C_IT_ERR, 170
 - I2C_IT_EVT, 170
 - I2C_IT_OVR, 170
 - I2C_IT_PECERR, 171
 - I2C_IT_RXNE, 171
 - I2C_IT_SB, 171
 - I2C_IT_SMBALERT, 171
 - I2C_IT_STOPF, 171
 - I2C_IT_TIMEOUT, 171
 - I2C_IT_TXE, 171
 - IS_I2C_CLEAR_IT, 171
 - IS_I2C_CONFIG_IT, 171
 - IS_I2C_GET_IT, 171
- I2C_IT_ADD10
 - I2C_interrupts_definition, 170
- I2C_IT_ADDR
 - I2C_interrupts_definition, 170
- I2C_IT_AF
 - I2C_interrupts_definition, 170
- I2C_IT_ARLO
 - I2C_interrupts_definition, 170
- I2C_IT_BERR
 - I2C_interrupts_definition, 170
- I2C_IT_BTFF
 - I2C_interrupts_definition, 170
- I2C_IT_BUF
 - I2C_interrupts_definition, 170
- I2C_IT_ERR
 - I2C_interrupts_definition, 170
- I2C_IT_EVT
 - I2C_interrupts_definition, 170
- I2C_IT_OVR
 - I2C_interrupts_definition, 170
- I2C_IT_PECERR
 - I2C_interrupts_definition, 171
- I2C_IT_RXNE
 - I2C_interrupts_definition, 171
- I2C_IT_SB
 - I2C_interrupts_definition, 171
- I2C_IT_SMBALERT
 - I2C_interrupts_definition, 171
- I2C_IT_STOPF
 - I2C_interrupts_definition, 171
- I2C_IT_TIMEOUT
 - I2C_interrupts_definition, 171
- I2C_IT_TXE
 - I2C_interrupts_definition, 171
- I2C_ITConfig
 - I2C, 134
 - Interrupts events and flags management functions, 161
- I2C_Mode
 - I2C_InitTypeDef, 318

- I2C_mode, 162
 - I2C_Mode_I2C, 163
 - I2C_Mode_SMBusDevice, 163
 - I2C_Mode_SMBusHost, 163
 - IS_I2C_MODE, 163
- I2C_Mode_I2C
 - I2C_mode, 163
- I2C_Mode_SMBusDevice
 - I2C_mode, 163
- I2C_Mode_SMBusHost
 - I2C_mode, 163
- I2C_NACK_position, 167
 - I2C_NACKPosition_Current, 167
 - I2C_NACKPosition_Next, 167
 - IS_I2C_NACK_POSITION, 168
- I2C_NACKPosition_Current
 - I2C_NACK_position, 167
- I2C_NACKPosition_Next
 - I2C_NACK_position, 167
- I2C_NACKPositionConfig
 - I2C, 135
 - Initialization and Configuration functions, 145
- I2C_own_address1, 179
 - IS_I2C_OWN_ADDRESS1, 179
- I2C_OwnAddress1
 - I2C_InitTypeDef, 318
- I2C_OwnAddress2Config
 - I2C, 136
 - Initialization and Configuration functions, 146
- I2C_PEC_position, 168
 - I2C_PECPosition_Current, 169
 - I2C_PECPosition_Next, 169
 - IS_I2C_PEC_POSITION, 169
- I2C_PECPosition_Current
 - I2C_PEC_position, 169
- I2C_PECPosition_Next
 - I2C_PEC_position, 169
- I2C_PECPositionConfig
 - I2C, 136
 - PEC management functions, 151
- I2C_Private_Functions, 140
- i2c_read
 - i2c.c, 349
 - i2c.h, 350
- I2C_ReadRegister
 - I2C, 137
 - Interrupts events and flags management functions, 161
- I2C_ReceiveData
 - Data transfers functions, 149
 - I2C, 137
- I2C_Register_CCR
 - I2C_registers, 166
- I2C_Register_CR1
 - I2C_registers, 166
- I2C_Register_CR2
 - I2C_registers, 166
- I2C_Register_DR
 - I2C_registers, 166
- I2C_Register_OAR1
 - I2C_registers, 166
- I2C_Register_OAR2
 - I2C_registers, 167
- I2C_Register_SR1
 - I2C_registers, 167
- I2C_Register_SR2
 - I2C_registers, 167
- I2C_Register_TRISE
 - I2C_registers, 167
- I2C_registers, 166
 - I2C_Register_CCR, 166
 - I2C_Register_CR1, 166
 - I2C_Register_CR2, 166
 - I2C_Register_DR, 166
 - I2C_Register_OAR1, 166
 - I2C_Register_OAR2, 167
 - I2C_Register_SR1, 167
 - I2C_Register_SR2, 167
 - I2C_Register_TRISE, 167
 - IS_I2C_REGISTER, 167
- I2C_Send7bitAddress
 - I2C, 138
 - Initialization and Configuration functions, 147
- I2C_SendData
 - Data transfers functions, 149
 - I2C, 138
- I2C_SMBus_alert_pin_level, 168
 - I2C_SMBusAlert_High, 168
 - I2C_SMBusAlert_Low, 168
 - IS_I2C_SMBUS_ALERT, 168
- I2C_SMBusAlert_High
 - I2C_SMBus_alert_pin_level, 168
- I2C_SMBusAlert_Low
 - I2C_SMBus_alert_pin_level, 168
- I2C_SMBusAlertConfig
 - I2C, 138
 - Initialization and Configuration functions, 147
- I2C_SoftwareResetCmd
 - I2C, 139
 - Initialization and Configuration functions, 147
- I2C_StretchClockCmd
 - I2C, 139
 - Initialization and Configuration functions, 148
- I2C_StructInit
 - I2C, 140
 - Initialization and Configuration functions, 148
- I2C_transfer_direction, 165
 - I2C_Direction_Receiver, 165
 - I2C_Direction_Transmitter, 165
 - IS_I2C_DIRECTION, 165
- I2C_TransmitPEC
 - I2C, 140
 - PEC management functions, 152
- i2c_write
 - i2c.c, 349
 - i2c.h, 350

- I2SSRC_BitNumber
 - RCC, [184](#)
- Init
 - ADC_HandleTypeDef, [314](#)
- Initialization and Configuration, [99](#)
 - GPIO_DeInit, [100](#)
 - GPIO_Init, [100](#)
 - GPIO_PinLockConfig, [101](#)
 - GPIO_StructInit, [101](#)
- Initialization and Configuration functions, [34](#), [141](#), [279](#)
 - ADC_Cmd, [35](#)
 - ADC_CommonInit, [35](#)
 - ADC_CommonStructInit, [35](#)
 - ADC_DeInit, [36](#)
 - ADC_Init, [36](#)
 - ADC_StructInit, [36](#)
 - I2C_AcknowledgeConfig, [142](#)
 - I2C_ARPCmd, [142](#)
 - I2C_Cmd, [142](#)
 - I2C_DeInit, [143](#)
 - I2C_DualAddressCmd, [143](#)
 - I2C_FastModeDutyCycleConfig, [143](#)
 - I2C_GeneralCallCmd, [144](#)
 - I2C_GenerateSTART, [144](#)
 - I2C_GenerateSTOP, [145](#)
 - I2C_Init, [145](#)
 - I2C_NACKPositionConfig, [145](#)
 - I2C_OwnAddress2Config, [146](#)
 - I2C_Send7bitAddress, [147](#)
 - I2C_SMBusAlertConfig, [147](#)
 - I2C_SoftwareResetCmd, [147](#)
 - I2C_StretchClockCmd, [148](#)
 - I2C_StructInit, [148](#)
 - USART_ClockInit, [280](#)
 - USART_ClockStructInit, [280](#)
 - USART_Cmd, [280](#)
 - USART_DeInit, [281](#)
 - USART_Init, [281](#)
 - USART_OneBitMethodCmd, [281](#)
 - USART_OverSampling8Cmd, [281](#)
 - USART_SetPrescaler, [282](#)
 - USART_StructInit, [282](#)
- Injected channels Configuration functions, [49](#)
 - ADC_AutoInjectedConvCmd, [50](#)
 - ADC_ExternalTrigInjectedConvConfig, [50](#)
 - ADC_ExternalTrigInjectedConvEdgeConfig, [51](#)
 - ADC_GetInjectedConversionValue, [52](#)
 - ADC_GetSoftwareStartInjectedConvCmdStatus, [52](#)
 - ADC_InjectedChannelConfig, [53](#)
 - ADC_InjectedDiscModeCmd, [55](#)
 - ADC_InjectedSequencerLengthConfig, [55](#)
 - ADC_SetInjectedOffset, [55](#)
 - ADC_SoftwareStartInjectedConv, [56](#)
- Input
 - gpio.h, [345](#)
- Instance
 - ADC_HandleTypeDef, [314](#)
- Internal and external clocks, PLL, CSS and MCO configuration functions, [212](#)
 - RCC_AdjustHSICalibrationValue, [213](#)
 - RCC_ClockSecuritySystemCmd, [214](#)
 - RCC_DeInit, [214](#)
 - RCC_HSEConfig, [215](#)
 - RCC_HSICmd, [215](#)
 - RCC_LSEConfig, [216](#)
 - RCC_LSICmd, [216](#)
 - RCC_MCO1Config, [217](#)
 - RCC_MCO2Config, [217](#)
 - RCC_PLLCmd, [218](#)
 - RCC_PLLConfig, [218](#)
 - RCC_PLLI2SCmd, [220](#)
 - RCC_PLLI2SConfig, [220](#)
 - RCC_WaitForHSEStartUp, [221](#)
- Interrupts and flags management functions, [56](#), [241](#), [292](#)
 - ADC_ClearFlag, [58](#)
 - ADC_ClearITPendingBit, [58](#)
 - ADC_GetFlagStatus, [59](#)
 - ADC_GetITStatus, [59](#)
 - ADC_ITConfig, [60](#)
 - RCC_ClearFlag, [241](#)
 - RCC_ClearITPendingBit, [242](#)
 - RCC_GetFlagStatus, [242](#)
 - RCC_GetITStatus, [243](#)
 - RCC_ITConfig, [244](#)
 - USART_ClearFlag, [294](#)
 - USART_ClearITPendingBit, [294](#)
 - USART_GetFlagStatus, [295](#)
 - USART_GetITStatus, [296](#)
 - USART_ITConfig, [296](#)
- Interrupts events and flags management functions, [153](#)
 - I2C_CheckEvent, [155](#)
 - I2C_ClearFlag, [156](#)
 - I2C_ClearITPendingBit, [157](#)
 - I2C_GetFlagStatus, [158](#)
 - I2C_GetITStatus, [159](#)
 - I2C_GetLastEvent, [160](#)
 - I2C_ITConfig, [161](#)
 - I2C_ReadRegister, [161](#)
- IrDA mode functions, [290](#)
 - USART_IrDACmd, [291](#)
 - USART_IrDAConfig, [291](#)
- IRQ_pin_index
 - gpio.c, [344](#)
- IRQ_port_num
 - gpio.c, [344](#)
- IRQ_status
 - gpio.c, [344](#)
- IS_ADC_ALL_PERIPH
 - ADC_Exported_Constants, [61](#)
- IS_ADC_ANALOG_WATCHDOG
 - ADC_analog_watchdog_selection, [82](#)
- IS_ADC_CHANNEL
 - ADC_channels, [76](#)
- IS_ADC_CLEAR_FLAG
 - ADC_flags_definition, [84](#)

IS_ADC_DATA_ALIGN
 ADC_data_align, 73
 IS_ADC_DMA_ACCESS_MODE
 ADC_Direct_memory_access_mode_for_multi_mode, 65
 IS_ADC_EXT_INJEC_TRIG
 ADC_extrenal_trigger_sources_for_injected_channels_conversion, 80
 IS_ADC_EXT_INJEC_TRIG_EDGE
 ADC_external_trigger_edge_for_injected_channels_conversion, 78
 IS_ADC_EXT_TRIG
 ADC_extrenal_trigger_sources_for_regular_channels_conversion, 72
 IS_ADC_EXT_TRIG_EDGE
 ADC_external_trigger_edge_for_regular_channels_conversion, 69
 IS_ADC_GET_FLAG
 ADC_flags_definition, 84
 IS_ADC_INJECTED_CHANNEL
 ADC_injected_channel_selection, 81
 IS_ADC_INJECTED_LENGTH
 ADC_injected_length, 86
 IS_ADC_INJECTED_RANK
 ADC_injected_rank, 86
 IS_ADC_IT
 ADC_interrupts_definition, 83
 IS_ADC_MODE
 ADC_Common_mode, 63
 IS_ADC_OFFSET
 ADC_injected_offset, 85
 IS_ADC_PRESCALER
 ADC_Prescaler, 64
 IS_ADC_REGULAR_DISC_NUMBER
 ADC_regular_discontinuous_mode_number, 87
 IS_ADC_REGULAR_LENGTH
 ADC_regular_length, 86
 IS_ADC_REGULAR_RANK
 ADC_regular_rank, 87
 IS_ADC_RESOLUTION
 ADC_resolution, 69
 IS_ADC_SAMPLE_TIME
 ADC_sampling_times, 77
 IS_ADC_SAMPLING_DELAY
 ADC_delay_between_2_sampling_phases, 68
 IS_ADC_THRESHOLD
 ADC_thresholds, 85
 IS_GET_GPIO_PIN
 GPIO_pins_define, 112
 IS_GPIO_AF
 GPIO_Alternat_function_selection_define, 121
 IS_GPIO_ALL_PERIPH
 GPIO, 89
 IS_GPIO_BIT_ACTION
 GPIO, 89
 IS_GPIO_MODE
 GPIO, 89
 IS_GPIO_OTYPE
 GPIO, 89
 IS_GPIO_PIN
 GPIO_pins_define, 112
 IS_GPIO_PIN_SOURCE
 GPIO_Pin_sources, 115
 IS_GPIO_PUPD
 GPIO, 89
 IS_GPIO_SPEED
 GPIO, 89
 IS_I2C_ACK_STATE
 I2C_acknowledgement, 164
 IS_I2C_ACKNOWLEDGE_ADDRESS
 I2C_acknowledged_address, 166
 IS_I2C_ALL_PERIPH
 I2C_Exported_Constants, 162
 IS_I2C_CLEAR_FLAG
 I2C_flags_definition, 174
 IS_I2C_CLEAR_IT
 I2C_interrupts_definition, 171
 IS_I2C_CLOCK_SPEED
 I2C_clock_speed, 179
 IS_I2C_CONFIG_IT
 I2C_interrupts_definition, 171
 IS_I2C_DIRECTION
 I2C_transfer_direction, 165
 IS_I2C_DUTY_CYCLE
 I2C_duty_cycle_in_fast_mode, 164
 IS_I2C_EVENT
 I2C_Events, 179
 IS_I2C_GET_FLAG
 I2C_flags_definition, 175
 IS_I2C_GET_IT
 I2C_interrupts_definition, 171
 IS_I2C_MODE
 I2C_mode, 163
 IS_I2C_NACK_POSITION
 I2C_NACK_position, 168
 IS_I2C_OWN_ADDRESS1
 I2C_own_address1, 179
 IS_I2C_PEC_POSITION
 I2C_PEC_position, 169
 IS_I2C_REGISTER
 I2C_registers, 167
 IS_I2C_SMBUS_ALERT
 I2C_SMBus_alert_pin_level, 168
 IS_RCC_AHB1_CLOCK_PERIPH
 RCC_AHB1_Peripherals, 254
 IS_RCC_AHB1_LPMODE_PERIPH
 RCC_AHB1_Peripherals, 254
 IS_RCC_AHB1_RESET_PERIPH
 RCC_AHB1_Peripherals, 254
 IS_RCC_AHB2_PERIPH
 RCC_AHB2_Peripherals, 256
 IS_RCC_AHB3_PERIPH
 RCC_AHB3_Peripherals, 257
 IS_RCC_APB1_PERIPH
 RCC_APB1_Peripherals, 257
 IS_RCC_APB2_PERIPH

- RCC_APB2_Peripherals, [259](#)
- IS_RCC_APB2_RESET_PERIPH
 - RCC_APB2_Peripherals, [259](#)
- IS_RCC_CALIBRATION_VALUE
 - RCC_Flag, [263](#)
- IS_RCC_CLEAR_IT
 - RCC_Interrupt_Source, [249](#)
- IS_RCC_FLAG
 - RCC_Flag, [263](#)
- IS_RCC_GET_IT
 - RCC_Interrupt_Source, [249](#)
- IS_RCC_HCLK
 - RCC_AHB_Clock_Source, [247](#)
- IS_RCC_HSE
 - RCC_HSE_configuration, [245](#)
- IS_RCC_I2SCLK_SOURCE
 - RCC_I2S_Clock_Source, [253](#)
- IS_RCC_IT
 - RCC_Interrupt_Source, [249](#)
- IS_RCC_LSE
 - RCC_LSE_Configuration, [250](#)
- IS_RCC_MCO1DIV
 - RCC_MCO1_Clock_Source_Prescaler, [261](#)
- IS_RCC_MCO1SOURCE
 - RCC_MCO1_Clock_Source_Prescaler, [261](#)
- IS_RCC_MCO2DIV
 - RCC_MCO2_Clock_Source_Prescaler, [262](#)
- IS_RCC_MCO2SOURCE
 - RCC_MCO2_Clock_Source_Prescaler, [262](#)
- IS_RCC_PCLK
 - RCC_APB1_APB2_Clock_Source, [248](#)
- IS_RCC_PLL_SOURCE
 - RCC_PLL_Clock_Source, [245](#)
- IS_RCC_PLLI2SN_VALUE
 - RCC_PLL_Clock_Source, [245](#)
- IS_RCC_PLLI2SR_VALUE
 - RCC_PLL_Clock_Source, [246](#)
- IS_RCC_PLLM_VALUE
 - RCC_PLL_Clock_Source, [246](#)
- IS_RCC_PLLN_VALUE
 - RCC_PLL_Clock_Source, [246](#)
- IS_RCC_PLLP_VALUE
 - RCC_PLL_Clock_Source, [246](#)
- IS_RCC_PLLQ_VALUE
 - RCC_PLL_Clock_Source, [246](#)
- IS_RCC_RTCCLK_SOURCE
 - RCC_RTC_Clock_Source, [251](#)
- IS_RCC_SYSCCLK_SOURCE
 - RCC_System_Clock_Source, [246](#)
- IS_USART_1236_PERIPH
 - USART_Exported_Constants, [297](#)
- IS_USART_ADDRESS
 - USART_Flags, [306](#)
- IS_USART_ALL_PERIPH
 - USART_Exported_Constants, [298](#)
- IS_USART_BAUDRATE
 - USART_Flags, [306](#)
- IS_USART_CLEAR_FLAG
 - USART_Flags, [306](#)
- IS_USART_CLEAR_IT
 - USART_Interrupt_definition, [303](#)
- IS_USART_CLOCK
 - USART_Clock, [301](#)
- IS_USART_CONFIG_IT
 - USART_Interrupt_definition, [303](#)
- IS_USART_CPHA
 - USART_Clock_Phase, [301](#)
- IS_USART_CPOL
 - USART_Clock_Polarity, [301](#)
- IS_USART_DATA
 - USART_Flags, [306](#)
- IS_USART_DMAREQ
 - USART_DMA_Requests, [304](#)
- IS_USART_FLAG
 - USART_Flags, [307](#)
- IS_USART_GET_IT
 - USART_Interrupt_definition, [303](#)
- IS_USART_HARDWARE_FLOW_CONTROL
 - USART_Hardware_Flow_Control, [300](#)
- IS_USART_IRDA_MODE
 - USART_IrDA_Low_Power, [306](#)
- IS_USART_LASTBIT
 - USART_Last_Bit, [302](#)
- IS_USART_LIN_BREAK_DETECT_LENGTH
 - USART_LIN_Break_Detection_Length, [305](#)
- IS_USART_MODE
 - USART_Mode, [300](#)
- IS_USART_PARITY
 - USART_Parity, [299](#)
- IS_USART_STOPBITS
 - USART_Stop_Bits, [298](#)
- IS_USART_WAKEUP
 - USART_WakeUp_methods, [305](#)
- IS_USART_WORD_LENGTH
 - USART_Word_Length, [298](#)
- IT_MASK
 - USART, [267](#)
- ITEN_MASK
 - I2C, [124](#)
- JDR_OFFSET
 - ADC, [13](#)
- JSQR_JL_RESET
 - ADC, [13](#)
- JSQR_JL_SET
 - ADC, [13](#)
- JSQR_JSQ_SET
 - ADC, [13](#)
- LED_OFF
 - platform.h, [352](#)
- LED_ON
 - platform.h, [353](#)
- LIN mode functions, [285](#)
 - USART_LINBreakDetectLengthConfig, [286](#)
 - USART_LINCmd, [287](#)
 - USART_SendBreak, [287](#)

Lock
 ADC_HandleTypeDef, 314
 LSION_BitNumber
 RCC, 184

 main
 main.c, 424
 main.c
 main, 424
 my_sqrt_fixed_point, 424
 my_sqrt_int, 426
 Mode
 GPIO_InitTypeDef, 317
 MultiProcessor Communication functions, 284
 USART_ReceiverWakeUpCmd, 284
 USART_SetAddress, 285
 USART_WakeUpConfig, 285
 my_sqrt_fixed_point
 main.c, 424
 my_sqrt_int
 main.c, 426

 NbrOfConversion
 _ADC_InitTypeDef, 312
 NbrOfCurrentConversionRank
 ADC_HandleTypeDef, 314
 NbrOfDiscConversion
 _ADC_InitTypeDef, 312
 NC
 platform.h, 356
 None
 gpio.h, 346

 Objects/adc.d, 422
 Objects/comparator.d, 422
 Objects/gpio.d, 422
 Objects/i2c.d, 422
 Objects/main.d, 422
 Objects/startup_stm32f401xe.d, 422
 Objects/stm32f4xx_adc.d, 422
 Objects/stm32f4xx_gpio.d, 422
 Objects/stm32f4xx_i2c.d, 422
 Objects/stm32f4xx_rcc.d, 422
 Objects/stm32f4xx_usart.d, 422
 Objects/system_stm32f4xx.d, 422
 Objects/timer.d, 422
 Objects/uart.d, 422
 Offset
 ADC_ChannelConfTypeDef, 313
 Output
 gpio.h, 346

 P_ADC
 platform.h, 353
 P_CMP_NEG
 platform.h, 353
 P_CMP_PLUS
 platform.h, 353
 P_DBG_ISR
 platform.h, 353
 P_DBG_MAIN
 platform.h, 353
 P_IR
 platform.h, 353
 P_LCD_DATA4
 platform.h, 353
 P_LCD_DATA5
 platform.h, 353
 P_LCD_DATA6
 platform.h, 353
 P_LCD_DATA7
 platform.h, 353
 P_LCD_E
 platform.h, 353
 P_LCD_RS
 platform.h, 353
 P_LCD_RW
 platform.h, 353
 P_LED_B
 platform.h, 353
 P_LED_G
 platform.h, 354
 P_LED_R
 platform.h, 354
 P_PERIOD
 platform.h, 354
 P_RX
 platform.h, 354
 P_SAMPLE
 platform.h, 354
 P_SCL
 platform.h, 354
 P_SDA
 platform.h, 354
 P_SPEAKER
 platform.h, 354
 P_SW
 platform.h, 354
 P_SW_CR
 platform.h, 354
 P_SW_DN
 platform.h, 354
 P_SW_LT
 platform.h, 354
 P_SW_RT
 platform.h, 354
 P_SW_UP
 platform.h, 354
 P_TX
 platform.h, 354
 PA_0
 platform.h, 355
 PA_1
 platform.h, 355
 PA_10
 platform.h, 355
 PA_11

- platform.h, [355](#)
- PA_12
 - platform.h, [355](#)
- PA_13
 - platform.h, [355](#)
- PA_14
 - platform.h, [355](#)
- PA_15
 - platform.h, [355](#)
- PA_2
 - platform.h, [355](#)
- PA_3
 - platform.h, [355](#)
- PA_4
 - platform.h, [355](#)
- PA_5
 - platform.h, [355](#)
- PA_6
 - platform.h, [355](#)
- PA_7
 - platform.h, [355](#)
- PA_8
 - platform.h, [355](#)
- PA_9
 - platform.h, [355](#)
- PB_0
 - platform.h, [355](#)
- PB_1
 - platform.h, [355](#)
- PB_10
 - platform.h, [355](#)
- PB_12
 - platform.h, [355](#)
- PB_13
 - platform.h, [355](#)
- PB_14
 - platform.h, [355](#)
- PB_15
 - platform.h, [355](#)
- PB_2
 - platform.h, [355](#)
- PB_3
 - platform.h, [355](#)
- PB_4
 - platform.h, [355](#)
- PB_5
 - platform.h, [355](#)
- PB_6
 - platform.h, [355](#)
- PB_7
 - platform.h, [355](#)
- PB_8
 - platform.h, [355](#)
- PB_9
 - platform.h, [355](#)
- PC_0
 - platform.h, [355](#)
- PC_1
 - platform.h, [355](#)
- PC_10
 - platform.h, [355](#)
- PC_11
 - platform.h, [356](#)
- PC_12
 - platform.h, [356](#)
- PC_13
 - platform.h, [356](#)
- PC_14
 - platform.h, [356](#)
- PC_15
 - platform.h, [356](#)
- PC_2
 - platform.h, [355](#)
- PC_3
 - platform.h, [355](#)
- PC_4
 - platform.h, [355](#)
- PC_5
 - platform.h, [355](#)
- PC_6
 - platform.h, [355](#)
- PC_7
 - platform.h, [355](#)
- PC_8
 - platform.h, [355](#)
- PC_9
 - platform.h, [355](#)
- PCLK1_Frequency
 - RCC_ClocksTypeDef, [319](#)
- PCLK2_Frequency
 - RCC_ClocksTypeDef, [319](#)
- PD_2
 - platform.h, [356](#)
- PEC management functions, [150](#)
 - I2C_CalculatePEC, [150](#)
 - I2C_GetPEC, [151](#)
 - I2C_PECPositionConfig, [151](#)
 - I2C_TransmitPEC, [152](#)
- peripheral
 - PinMap, [319](#)
- Peripheral clocks configuration functions, [226](#)
 - RCC_AHB1PeriphClockCmd, [227](#)
 - RCC_AHB1PeriphClockLPModeCmd, [228](#)
 - RCC_AHB1PeriphResetCmd, [229](#)
 - RCC_AHB2PeriphClockCmd, [230](#)
 - RCC_AHB2PeriphClockLPModeCmd, [231](#)
 - RCC_AHB2PeriphResetCmd, [231](#)
 - RCC_AHB3PeriphClockCmd, [232](#)
 - RCC_AHB3PeriphClockLPModeCmd, [232](#)
 - RCC_AHB3PeriphResetCmd, [233](#)
 - RCC_APB1PeriphClockCmd, [233](#)
 - RCC_APB1PeriphClockLPModeCmd, [234](#)
 - RCC_APB1PeriphResetCmd, [235](#)
 - RCC_APB2PeriphClockCmd, [236](#)
 - RCC_APB2PeriphClockLPModeCmd, [237](#)
 - RCC_APB2PeriphResetCmd, [238](#)

- RCC_BackupResetCmd, [239](#)
- RCC_I2SCLKConfig, [239](#)
- RCC_RTCCLKCmd, [240](#)
- RCC_RTCCLKConfig, [240](#)
- PH_0
 - platform.h, [356](#)
- PH_1
 - platform.h, [356](#)
- Pin
 - GPIO_InitTypeDef, [317](#)
 - platform.h, [355](#)
- pin
 - analogin_s, [316](#)
 - PinMap, [319](#)
- pin_function
 - adc.c, [324](#)
 - adc.h, [335](#)
- PinMap, [319](#)
 - function, [319](#)
 - peripheral, [319](#)
 - pin, [319](#)
- PinMap_ADC
 - adc.c, [325](#)
- pinmap_find_function
 - adc.c, [324](#)
- pinmap_find_peripheral
 - adc.c, [325](#)
 - adc.h, [335](#)
- pinmap_function
 - adc.c, [325](#)
 - adc.h, [335](#)
- pinmap_peripheral
 - adc.c, [325](#)
 - adc.h, [335](#)
- pinmap_pinout
 - adc.c, [325](#)
 - adc.h, [336](#)
- PinMode
 - gpio.h, [345](#)
- platform.h
 - ADC_BITS, [352](#)
 - ADC_MASK, [352](#)
 - CLK_FREQ, [352](#)
 - DAC_BITS, [352](#)
 - DAC_MASK, [352](#)
 - GET_PIN_INDEX, [352](#)
 - GET_PORT, [352](#)
 - GET_PORT_INDEX, [352](#)
 - LED_OFF, [352](#)
 - LED_ON, [353](#)
 - NC, [356](#)
 - P_ADC, [353](#)
 - P_CMP_NEG, [353](#)
 - P_CMP_PLUS, [353](#)
 - P_DBG_ISR, [353](#)
 - P_DBG_MAIN, [353](#)
 - P_IR, [353](#)
 - P_LCD_DATA4, [353](#)
 - P_LCD_DATA5, [353](#)
 - P_LCD_DATA6, [353](#)
 - P_LCD_DATA7, [353](#)
 - P_LCD_E, [353](#)
 - P_LCD_RS, [353](#)
 - P_LCD_RW, [353](#)
 - P_LED_B, [353](#)
 - P_LED_G, [354](#)
 - P_LED_R, [354](#)
 - P_PERIOD, [354](#)
 - P_RX, [354](#)
 - P_SAMPLE, [354](#)
 - P_SCL, [354](#)
 - P_SDA, [354](#)
 - P_SPEAKER, [354](#)
 - P_SW, [354](#)
 - P_SW_CR, [354](#)
 - P_SW_DN, [354](#)
 - P_SW_LT, [354](#)
 - P_SW_RT, [354](#)
 - P_SW_UP, [354](#)
 - P_TX, [354](#)
 - PA_0, [355](#)
 - PA_1, [355](#)
 - PA_10, [355](#)
 - PA_11, [355](#)
 - PA_12, [355](#)
 - PA_13, [355](#)
 - PA_14, [355](#)
 - PA_15, [355](#)
 - PA_2, [355](#)
 - PA_3, [355](#)
 - PA_4, [355](#)
 - PA_5, [355](#)
 - PA_6, [355](#)
 - PA_7, [355](#)
 - PA_8, [355](#)
 - PA_9, [355](#)
 - PB_0, [355](#)
 - PB_1, [355](#)
 - PB_10, [355](#)
 - PB_12, [355](#)
 - PB_13, [355](#)
 - PB_14, [355](#)
 - PB_15, [355](#)
 - PB_2, [355](#)
 - PB_3, [355](#)
 - PB_4, [355](#)
 - PB_5, [355](#)
 - PB_6, [355](#)
 - PB_7, [355](#)
 - PB_8, [355](#)
 - PB_9, [355](#)
 - PC_0, [355](#)
 - PC_1, [355](#)
 - PC_10, [355](#)
 - PC_11, [356](#)
 - PC_12, [356](#)

- PC_13, [356](#)
- PC_14, [356](#)
- PC_15, [356](#)
- PC_2, [355](#)
- PC_3, [355](#)
- PC_4, [355](#)
- PC_5, [355](#)
- PC_6, [355](#)
- PC_7, [355](#)
- PC_8, [355](#)
- PC_9, [355](#)
- PD_2, [356](#)
- PH_0, [356](#)
- PH_1, [356](#)
- Pin, [355](#)
- PLLI2SON_BitNumber
 - RCC, [184](#)
- PLLON_BitNumber
 - RCC, [184](#)
- PortA
 - adc.h, [334](#)
- PortB
 - adc.h, [334](#)
- PortC
 - adc.h, [334](#)
- PortD
 - adc.h, [334](#)
- PortE
 - adc.h, [334](#)
- PortH
 - adc.h, [334](#)
- PortName
 - adc.h, [334](#)
- priority
 - gpio.c, [344](#)
- prioritygroup
 - gpio.c, [344](#)
- Pull
 - GPIO_InitTypeDef, [317](#)
- PullDown
 - gpio.h, [346](#)
- PullUp
 - gpio.h, [346](#)
- Rank
 - ADC_ChannelConfTypeDef, [313](#)
- RCC, [179](#)
 - BDCR_ADDRESS, [182](#)
 - BDCR_BDRST_BB, [182](#)
 - BDCR_OFFSET, [182](#)
 - BDCR_RTCEN_BB, [182](#)
 - BDRST_BitNumber, [182](#)
 - CFGR_I2SSRC_BB, [182](#)
 - CFGR_MCO1_RESET_MASK, [183](#)
 - CFGR_MCO2_RESET_MASK, [183](#)
 - CFGR_OFFSET, [183](#)
 - CIR_BYTE2_ADDRESS, [183](#)
 - CIR_BYTE3_ADDRESS, [183](#)
 - CR_BYTE3_ADDRESS, [183](#)
 - CR_CSSON_BB, [183](#)
 - CR_HSION_BB, [183](#)
 - CR_OFFSET, [183](#)
 - CR_PLLI2SON_BB, [183](#)
 - CR_PLLON_BB, [183](#)
 - CSR_LSION_BB, [183](#)
 - CSR_OFFSET, [183](#)
 - CSSON_BitNumber, [183](#)
 - FLAG_MASK, [183](#)
 - HSION_BitNumber, [184](#)
 - I2SSRC_BitNumber, [184](#)
 - LSION_BitNumber, [184](#)
 - PLLI2SON_BitNumber, [184](#)
 - PLLON_BitNumber, [184](#)
 - RCC_AdjustHSICalibrationValue, [184](#)
 - RCC_AHB1PeriphClockCmd, [184](#)
 - RCC_AHB1PeriphClockLPModeCmd, [185](#)
 - RCC_AHB1PeriphResetCmd, [186](#)
 - RCC_AHB2PeriphClockCmd, [187](#)
 - RCC_AHB2PeriphClockLPModeCmd, [188](#)
 - RCC_AHB2PeriphResetCmd, [188](#)
 - RCC_AHB3PeriphClockCmd, [189](#)
 - RCC_AHB3PeriphClockLPModeCmd, [189](#)
 - RCC_AHB3PeriphResetCmd, [190](#)
 - RCC_APB1PeriphClockCmd, [190](#)
 - RCC_APB1PeriphClockLPModeCmd, [191](#)
 - RCC_APB1PeriphResetCmd, [192](#)
 - RCC_APB2PeriphClockCmd, [193](#)
 - RCC_APB2PeriphClockLPModeCmd, [194](#)
 - RCC_APB2PeriphResetCmd, [195](#)
 - RCC_BackupResetCmd, [196](#)
 - RCC_ClearFlag, [196](#)
 - RCC_ClearITPendingBit, [197](#)
 - RCC_ClockSecuritySystemCmd, [197](#)
 - RCC_DeInit, [198](#)
 - RCC_GetClocksFreq, [198](#)
 - RCC_GetFlagStatus, [199](#)
 - RCC_GetITStatus, [200](#)
 - RCC_GetSYSCLKSource, [200](#)
 - RCC_HCLKConfig, [200](#)
 - RCC_HSEConfig, [201](#)
 - RCC_HSICmd, [202](#)
 - RCC_I2SCLKConfig, [202](#)
 - RCC_ITConfig, [203](#)
 - RCC_LSEConfig, [203](#)
 - RCC_LSIConfig, [204](#)
 - RCC_MCO1Config, [204](#)
 - RCC_MCO2Config, [205](#)
 - RCC_OFFSET, [184](#)
 - RCC_PCLK1Config, [206](#)
 - RCC_PCLK2Config, [206](#)
 - RCC_PLLCmd, [207](#)
 - RCC_PLLConfig, [207](#)
 - RCC_PLLI2SCmd, [209](#)
 - RCC_PLLI2SConfig, [209](#)
 - RCC_RTCCLKCmd, [210](#)
 - RCC_RTCCLKConfig, [210](#)
 - RCC_SYSCLKConfig, [211](#)

- RCC_WaitForHSEStartUp, [211](#)
- RTGEN_BitNumber, [184](#)
- RCC_ADC1_CLK_ENABLE
 - adc.h, [332](#)
- RCC_AdjustHSICalibrationValue
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [213](#)
 - RCC, [184](#)
- RCC_AHB1_Peripherals, [254](#)
 - IS_RCC_AHB1_CLOCK_PERIPH, [254](#)
 - IS_RCC_AHB1_LPMODE_PERIPH, [254](#)
 - IS_RCC_AHB1_RESET_PERIPH, [254](#)
 - RCC_AHB1Periph_BKPSRAM, [254](#)
 - RCC_AHB1Periph_CCMDATARAMEN, [254](#)
 - RCC_AHB1Periph_CRC, [254](#)
 - RCC_AHB1Periph_DMA1, [255](#)
 - RCC_AHB1Periph_DMA2, [255](#)
 - RCC_AHB1Periph_ETH_MAC, [255](#)
 - RCC_AHB1Periph_ETH_MAC_PTP, [255](#)
 - RCC_AHB1Periph_ETH_MAC_Rx, [255](#)
 - RCC_AHB1Periph_ETH_MAC_Tx, [255](#)
 - RCC_AHB1Periph_FLITF, [255](#)
 - RCC_AHB1Periph_GPIOA, [255](#)
 - RCC_AHB1Periph_GPIOB, [255](#)
 - RCC_AHB1Periph_GPIOC, [255](#)
 - RCC_AHB1Periph_GPIOD, [255](#)
 - RCC_AHB1Periph_GPIOE, [255](#)
 - RCC_AHB1Periph_GPIOF, [255](#)
 - RCC_AHB1Periph_GPIOG, [255](#)
 - RCC_AHB1Periph_GPIOH, [255](#)
 - RCC_AHB1Periph_GPIOI, [256](#)
 - RCC_AHB1Periph_OTG_HS, [256](#)
 - RCC_AHB1Periph_OTG_HS_ULPI, [256](#)
 - RCC_AHB1Periph_SRAM1, [256](#)
 - RCC_AHB1Periph_SRAM2, [256](#)
- RCC_AHB1Periph_BKPSRAM
 - RCC_AHB1_Peripherals, [254](#)
- RCC_AHB1Periph_CCMDATARAMEN
 - RCC_AHB1_Peripherals, [254](#)
- RCC_AHB1Periph_CRC
 - RCC_AHB1_Peripherals, [254](#)
- RCC_AHB1Periph_DMA1
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_DMA2
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_ETH_MAC
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_ETH_MAC_PTP
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_ETH_MAC_Rx
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_ETH_MAC_Tx
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_FLITF
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_GPIOA
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_GPIOB
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_GPIOC
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_GPIOD
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_GPIOE
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_GPIOF
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_GPIOG
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_GPIOH
 - RCC_AHB1_Peripherals, [255](#)
- RCC_AHB1Periph_GPIOI
 - RCC_AHB1_Peripherals, [256](#)
- RCC_AHB1Periph_OTG_HS
 - RCC_AHB1_Peripherals, [256](#)
- RCC_AHB1Periph_OTG_HS_ULPI
 - RCC_AHB1_Peripherals, [256](#)
- RCC_AHB1Periph_SRAM1
 - RCC_AHB1_Peripherals, [256](#)
- RCC_AHB1Periph_SRAM2
 - RCC_AHB1_Peripherals, [256](#)
- RCC_AHB1PeriphClockCmd
 - Peripheral clocks configuration functions, [227](#)
 - RCC, [184](#)
- RCC_AHB1PeriphClockLPModeCmd
 - Peripheral clocks configuration functions, [228](#)
 - RCC, [185](#)
- RCC_AHB1PeriphResetCmd
 - Peripheral clocks configuration functions, [229](#)
 - RCC, [186](#)
- RCC_AHB2_Peripherals, [256](#)
 - IS_RCC_AHB2_PERIPH, [256](#)
 - RCC_AHB2Periph_Cryp, [256](#)
 - RCC_AHB2Periph_DCMI, [256](#)
 - RCC_AHB2Periph_HASH, [256](#)
 - RCC_AHB2Periph_OTG_FS, [256](#)
 - RCC_AHB2Periph_RNG, [256](#)
- RCC_AHB2Periph_Cryp
 - RCC_AHB2_Peripherals, [256](#)
- RCC_AHB2Periph_DCMI
 - RCC_AHB2_Peripherals, [256](#)
- RCC_AHB2Periph_HASH
 - RCC_AHB2_Peripherals, [256](#)
- RCC_AHB2Periph_OTG_FS
 - RCC_AHB2_Peripherals, [256](#)
- RCC_AHB2Periph_RNG
 - RCC_AHB2_Peripherals, [256](#)
- RCC_AHB2PeriphClockCmd
 - Peripheral clocks configuration functions, [230](#)
 - RCC, [187](#)
- RCC_AHB2PeriphClockLPModeCmd
 - Peripheral clocks configuration functions, [231](#)
 - RCC, [188](#)
- RCC_AHB2PeriphResetCmd
 - Peripheral clocks configuration functions, [231](#)
 - RCC, [188](#)

- RCC_AHB3_Peripherals, [257](#)
 - IS_RCC_AHB3_PERIPH, [257](#)
 - RCC_AHB3Periph_FSMC, [257](#)
- RCC_AHB3Periph_FSMC
 - RCC_AHB3_Peripherals, [257](#)
- RCC_AHB3PeriphClockCmd
 - Peripheral clocks configuration functions, [232](#)
 - RCC, [189](#)
- RCC_AHB3PeriphClockLPModeCmd
 - Peripheral clocks configuration functions, [232](#)
 - RCC, [189](#)
- RCC_AHB3PeriphResetCmd
 - Peripheral clocks configuration functions, [233](#)
 - RCC, [190](#)
- RCC_AHB_Clock_Source, [247](#)
 - IS_RCC_HCLK, [247](#)
 - RCC_SYSCLK_Div1, [247](#)
 - RCC_SYSCLK_Div128, [247](#)
 - RCC_SYSCLK_Div16, [247](#)
 - RCC_SYSCLK_Div2, [247](#)
 - RCC_SYSCLK_Div256, [247](#)
 - RCC_SYSCLK_Div4, [247](#)
 - RCC_SYSCLK_Div512, [247](#)
 - RCC_SYSCLK_Div64, [248](#)
 - RCC_SYSCLK_Div8, [248](#)
- RCC_APB1_APB2_Clock_Source, [248](#)
 - IS_RCC_PCLK, [248](#)
 - RCC_HCLK_Div1, [248](#)
 - RCC_HCLK_Div16, [248](#)
 - RCC_HCLK_Div2, [248](#)
 - RCC_HCLK_Div4, [248](#)
 - RCC_HCLK_Div8, [248](#)
- RCC_APB1_Peripherals, [257](#)
 - IS_RCC_APB1_PERIPH, [257](#)
 - RCC_APB1Periph_CAN1, [257](#)
 - RCC_APB1Periph_CAN2, [258](#)
 - RCC_APB1Periph_DAC, [258](#)
 - RCC_APB1Periph_I2C1, [258](#)
 - RCC_APB1Periph_I2C2, [258](#)
 - RCC_APB1Periph_I2C3, [258](#)
 - RCC_APB1Periph_PWR, [258](#)
 - RCC_APB1Periph_SPI2, [258](#)
 - RCC_APB1Periph_SPI3, [258](#)
 - RCC_APB1Periph_TIM12, [258](#)
 - RCC_APB1Periph_TIM13, [258](#)
 - RCC_APB1Periph_TIM14, [258](#)
 - RCC_APB1Periph_TIM2, [258](#)
 - RCC_APB1Periph_TIM3, [258](#)
 - RCC_APB1Periph_TIM4, [258](#)
 - RCC_APB1Periph_TIM5, [258](#)
 - RCC_APB1Periph_TIM6, [259](#)
 - RCC_APB1Periph_TIM7, [259](#)
 - RCC_APB1Periph_UART4, [259](#)
 - RCC_APB1Periph_UART5, [259](#)
 - RCC_APB1Periph_USART2, [259](#)
 - RCC_APB1Periph_USART3, [259](#)
 - RCC_APB1Periph_WWDG, [259](#)
- RCC_APB1Periph_CAN1
 - RCC_APB1_Peripherals, [257](#)
- RCC_APB1Periph_CAN2
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_DAC
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_I2C1
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_I2C2
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_I2C3
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_PWR
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_SPI2
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_SPI3
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_TIM12
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_TIM13
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_TIM14
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_TIM2
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_TIM3
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_TIM4
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_TIM5
 - RCC_APB1_Peripherals, [258](#)
- RCC_APB1Periph_TIM6
 - RCC_APB1_Peripherals, [259](#)
- RCC_APB1Periph_TIM7
 - RCC_APB1_Peripherals, [259](#)
- RCC_APB1Periph_UART4
 - RCC_APB1_Peripherals, [259](#)
- RCC_APB1Periph_UART5
 - RCC_APB1_Peripherals, [259](#)
- RCC_APB1Periph_USART2
 - RCC_APB1_Peripherals, [259](#)
- RCC_APB1Periph_USART3
 - RCC_APB1_Peripherals, [259](#)
- RCC_APB1Periph_WWDG
 - RCC_APB1_Peripherals, [259](#)
- RCC_APB1PeriphClockCmd
 - Peripheral clocks configuration functions, [233](#)
 - RCC, [190](#)
- RCC_APB1PeriphClockLPModeCmd
 - Peripheral clocks configuration functions, [234](#)
 - RCC, [191](#)
- RCC_APB1PeriphResetCmd
 - Peripheral clocks configuration functions, [235](#)
 - RCC, [192](#)
- RCC_APB2_Peripherals, [259](#)
 - IS_RCC_APB2_PERIPH, [259](#)
 - IS_RCC_APB2_RESET_PERIPH, [259](#)
 - RCC_APB2Periph_ADC, [260](#)

- RCC_APB2Periph_ADC1, [260](#)
- RCC_APB2Periph_ADC2, [260](#)
- RCC_APB2Periph_ADC3, [260](#)
- RCC_APB2Periph_SDIO, [260](#)
- RCC_APB2Periph_SPI1, [260](#)
- RCC_APB2Periph_SYSCFG, [260](#)
- RCC_APB2Periph_TIM1, [260](#)
- RCC_APB2Periph_TIM10, [260](#)
- RCC_APB2Periph_TIM11, [260](#)
- RCC_APB2Periph_TIM8, [260](#)
- RCC_APB2Periph_TIM9, [260](#)
- RCC_APB2Periph_USART1, [260](#)
- RCC_APB2Periph_USART6, [260](#)
- RCC_APB2Periph_ADC
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_ADC1
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_ADC2
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_ADC3
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_SDIO
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_SPI1
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_SYSCFG
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_TIM1
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_TIM10
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_TIM11
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_TIM8
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_TIM9
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_USART1
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2Periph_USART6
 - RCC_APB2_Peripherals, [260](#)
- RCC_APB2PeriphClockCmd
 - Peripheral clocks configuration functions, [236](#)
 - RCC, [193](#)
- RCC_APB2PeriphClockLPMModeCmd
 - Peripheral clocks configuration functions, [237](#)
 - RCC, [194](#)
- RCC_APB2PeriphResetCmd
 - Peripheral clocks configuration functions, [238](#)
 - RCC, [195](#)
- RCC_BackupResetCmd
 - Peripheral clocks configuration functions, [239](#)
 - RCC, [196](#)
- RCC_ClearFlag
 - Interrupts and flags management functions, [241](#)
 - RCC, [196](#)
- RCC_ClearITPendingBit
 - Interrupts and flags management functions, [242](#)
- RCC, [197](#)
- RCC_ClockSecuritySystemCmd
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [214](#)
 - RCC, [197](#)
- RCC_ClocksTypeDef, [319](#)
 - HCLK_Frequency, [319](#)
 - PCLK1_Frequency, [319](#)
 - PCLK2_Frequency, [319](#)
 - SYSCLK_Frequency, [319](#)
- RCC_DeInit
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [214](#)
 - RCC, [198](#)
- RCC_Exported_Constants, [244](#)
- RCC_Flag, [263](#)
 - IS_RCC_CALIBRATION_VALUE, [263](#)
 - IS_RCC_FLAG, [263](#)
 - RCC_FLAG BORRST, [264](#)
 - RCC_FLAG HSERDY, [264](#)
 - RCC_FLAG HSIRDY, [264](#)
 - RCC_FLAG IWDGRST, [264](#)
 - RCC_FLAG LPWRRST, [264](#)
 - RCC_FLAG LSERDY, [264](#)
 - RCC_FLAG LSIRDY, [264](#)
 - RCC_FLAG PINRST, [264](#)
 - RCC_FLAG PLLI2SRDY, [264](#)
 - RCC_FLAG PLLRDY, [264](#)
 - RCC_FLAG PORRST, [264](#)
 - RCC_FLAG SFTRST, [264](#)
 - RCC_FLAG WWDGRST, [264](#)
- RCC_FLAG BORRST
 - RCC_Flag, [264](#)
- RCC_FLAG HSERDY
 - RCC_Flag, [264](#)
- RCC_FLAG HSIRDY
 - RCC_Flag, [264](#)
- RCC_FLAG IWDGRST
 - RCC_Flag, [264](#)
- RCC_FLAG LPWRRST
 - RCC_Flag, [264](#)
- RCC_FLAG LSERDY
 - RCC_Flag, [264](#)
- RCC_FLAG LSIRDY
 - RCC_Flag, [264](#)
- RCC_FLAG PINRST
 - RCC_Flag, [264](#)
- RCC_FLAG PLLI2SRDY
 - RCC_Flag, [264](#)
- RCC_FLAG PLLRDY
 - RCC_Flag, [264](#)
- RCC_FLAG PORRST
 - RCC_Flag, [264](#)
- RCC_FLAG SFTRST
 - RCC_Flag, [264](#)
- RCC_FLAG WWDGRST
 - RCC_Flag, [264](#)
- RCC_GetClocksFreq

- RCC, [198](#)
 - System AHB and APB busses clocks configuration functions, [222](#)
- RCC_GetFlagStatus
 - Interrupts and flags management functions, [242](#)
 - RCC, [199](#)
- RCC_GetITStatus
 - Interrupts and flags management functions, [243](#)
 - RCC, [200](#)
- RCC_GetSYSCLKSource
 - RCC, [200](#)
 - System AHB and APB busses clocks configuration functions, [223](#)
- RCC_GPIOA_CLK_ENABLE
 - adc.h, [332](#)
- RCC_GPIOB_CLK_ENABLE
 - adc.h, [332](#)
- RCC_GPIOC_CLK_ENABLE
 - adc.h, [332](#)
- RCC_HCLK_Div1
 - RCC_APB1_APB2_Clock_Source, [248](#)
- RCC_HCLK_Div16
 - RCC_APB1_APB2_Clock_Source, [248](#)
- RCC_HCLK_Div2
 - RCC_APB1_APB2_Clock_Source, [248](#)
- RCC_HCLK_Div4
 - RCC_APB1_APB2_Clock_Source, [248](#)
- RCC_HCLK_Div8
 - RCC_APB1_APB2_Clock_Source, [248](#)
- RCC_HCLKConfig
 - RCC, [200](#)
 - System AHB and APB busses clocks configuration functions, [223](#)
- RCC_HSE_Bypass
 - RCC_HSE_configuration, [245](#)
- RCC_HSE_configuration, [245](#)
 - IS_RCC_HSE, [245](#)
 - RCC_HSE_Bypass, [245](#)
 - RCC_HSE_OFF, [245](#)
 - RCC_HSE_ON, [245](#)
- RCC_HSE_OFF
 - RCC_HSE_configuration, [245](#)
- RCC_HSE_ON
 - RCC_HSE_configuration, [245](#)
- RCC_HSEConfig
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [215](#)
 - RCC, [201](#)
- RCC_HSICmd
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [215](#)
 - RCC, [202](#)
- RCC_I2S2CLKSource_Ext
 - RCC_I2S_Clock_Source, [253](#)
- RCC_I2S2CLKSource_PLLI2S
 - RCC_I2S_Clock_Source, [253](#)
- RCC_I2S_Clock_Source, [253](#)
 - IS_RCC_I2SCLK_SOURCE, [253](#)
- RCC_I2S2CLKSource_Ext, [253](#)
- RCC_I2S2CLKSource_PLLI2S, [253](#)
- RCC_I2SCLKConfig
 - Peripheral clocks configuration functions, [239](#)
 - RCC, [202](#)
- RCC_Interrupt_Source, [249](#)
 - IS_RCC_CLEAR_IT, [249](#)
 - IS_RCC_GET_IT, [249](#)
 - IS_RCC_IT, [249](#)
 - RCC_IT_CSS, [249](#)
 - RCC_IT_HSERDY, [249](#)
 - RCC_IT_HSIRDY, [249](#)
 - RCC_IT_LSERDY, [249](#)
 - RCC_IT_LSIRDY, [249](#)
 - RCC_IT_PLI2SRDY, [249](#)
 - RCC_IT_PLLRDY, [249](#)
- RCC_IT_CSS
 - RCC_Interrupt_Source, [249](#)
- RCC_IT_HSERDY
 - RCC_Interrupt_Source, [249](#)
- RCC_IT_HSIRDY
 - RCC_Interrupt_Source, [249](#)
- RCC_IT_LSERDY
 - RCC_Interrupt_Source, [249](#)
- RCC_IT_LSIRDY
 - RCC_Interrupt_Source, [249](#)
- RCC_IT_PLI2SRDY
 - RCC_Interrupt_Source, [249](#)
- RCC_IT_PLLRDY
 - RCC_Interrupt_Source, [249](#)
- RCC_ITConfig
 - Interrupts and flags management functions, [244](#)
 - RCC, [203](#)
- RCC_LSE_Bypass
 - RCC_LSE_Configuration, [250](#)
- RCC_LSE_Configuration, [250](#)
 - IS_RCC_LSE, [250](#)
 - RCC_LSE_Bypass, [250](#)
 - RCC_LSE_OFF, [250](#)
 - RCC_LSE_ON, [250](#)
- RCC_LSE_OFF
 - RCC_LSE_Configuration, [250](#)
- RCC_LSE_ON
 - RCC_LSE_Configuration, [250](#)
- RCC_LSEConfig
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [216](#)
 - RCC, [203](#)
- RCC_LSICmd
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [216](#)
 - RCC, [204](#)
- RCC_MCO1_Clock_Source_Prescaler, [261](#)
 - IS_RCC_MCO1DIV, [261](#)
 - IS_RCC_MCO1SOURCE, [261](#)
 - RCC_MCO1Div_1, [261](#)
 - RCC_MCO1Div_2, [261](#)
 - RCC_MCO1Div_3, [261](#)

- RCC_MCO1Div_4, [261](#)
- RCC_MCO1Div_5, [261](#)
- RCC_MCO1Source_HSE, [261](#)
- RCC_MCO1Source_HSI, [261](#)
- RCC_MCO1Source_LSE, [262](#)
- RCC_MCO1Source_PLLCLK, [262](#)
- RCC_MCO1Config
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [217](#)
 - RCC, [204](#)
- RCC_MCO1Div_1
 - RCC_MCO1_Clock_Source_Prescaler, [261](#)
- RCC_MCO1Div_2
 - RCC_MCO1_Clock_Source_Prescaler, [261](#)
- RCC_MCO1Div_3
 - RCC_MCO1_Clock_Source_Prescaler, [261](#)
- RCC_MCO1Div_4
 - RCC_MCO1_Clock_Source_Prescaler, [261](#)
- RCC_MCO1Div_5
 - RCC_MCO1_Clock_Source_Prescaler, [261](#)
- RCC_MCO1Source_HSE
 - RCC_MCO1_Clock_Source_Prescaler, [261](#)
- RCC_MCO1Source_HSI
 - RCC_MCO1_Clock_Source_Prescaler, [261](#)
- RCC_MCO1Source_LSE
 - RCC_MCO1_Clock_Source_Prescaler, [262](#)
- RCC_MCO1Source_PLLCLK
 - RCC_MCO1_Clock_Source_Prescaler, [262](#)
- RCC_MCO2_Clock_Source_Prescaler, [262](#)
 - IS_RCC_MCO2DIV, [262](#)
 - IS_RCC_MCO2SOURCE, [262](#)
 - RCC_MCO2Div_1, [262](#)
 - RCC_MCO2Div_2, [262](#)
 - RCC_MCO2Div_3, [262](#)
 - RCC_MCO2Div_4, [262](#)
 - RCC_MCO2Div_5, [263](#)
 - RCC_MCO2Source_HSE, [263](#)
 - RCC_MCO2Source_PLLCLK, [263](#)
 - RCC_MCO2Source_PLLI2SCLK, [263](#)
 - RCC_MCO2Source_SYSCLK, [263](#)
- RCC_MCO2Config
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [217](#)
 - RCC, [205](#)
- RCC_MCO2Div_1
 - RCC_MCO2_Clock_Source_Prescaler, [262](#)
- RCC_MCO2Div_2
 - RCC_MCO2_Clock_Source_Prescaler, [262](#)
- RCC_MCO2Div_3
 - RCC_MCO2_Clock_Source_Prescaler, [262](#)
- RCC_MCO2Div_4
 - RCC_MCO2_Clock_Source_Prescaler, [262](#)
- RCC_MCO2Div_5
 - RCC_MCO2_Clock_Source_Prescaler, [263](#)
- RCC_MCO2Source_HSE
 - RCC_MCO2_Clock_Source_Prescaler, [263](#)
- RCC_MCO2Source_PLLCLK
 - RCC_MCO2_Clock_Source_Prescaler, [263](#)
- RCC_MCO2Source_PLLI2SCLK
 - RCC_MCO2_Clock_Source_Prescaler, [263](#)
- RCC_MCO2Source_SYSCLK
 - RCC_MCO2_Clock_Source_Prescaler, [263](#)
- RCC_MCO2Source_PLLI2SCLK
 - RCC_MCO2_Clock_Source_Prescaler, [263](#)
- RCC_MCO2Source_SYSCLK
 - RCC_MCO2_Clock_Source_Prescaler, [263](#)
- RCC_OFFSET
 - RCC, [184](#)
- RCC_PCLK1Config
 - RCC, [206](#)
 - System AHB and APB busses clocks configuration functions, [224](#)
- RCC_PCLK2Config
 - RCC, [206](#)
 - System AHB and APB busses clocks configuration functions, [225](#)
- RCC_PLL_Clock_Source, [245](#)
 - IS_RCC_PLL_SOURCE, [245](#)
 - IS_RCC_PLLI2SN_VALUE, [245](#)
 - IS_RCC_PLLI2SR_VALUE, [246](#)
 - IS_RCC_PLLM_VALUE, [246](#)
 - IS_RCC_PLLN_VALUE, [246](#)
 - IS_RCC_PLLP_VALUE, [246](#)
 - IS_RCC_PLLQ_VALUE, [246](#)
 - RCC_PLLSource_HSE, [246](#)
 - RCC_PLLSource_HSI, [246](#)
- RCC_PLLCmd
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [218](#)
 - RCC, [207](#)
- RCC_PLLConfig
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [218](#)
 - RCC, [207](#)
- RCC_PLLI2SCmd
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [220](#)
 - RCC, [209](#)
- RCC_PLLI2SConfig
 - Internal and external clocks, PLL, CSS and MCO configuration functions, [220](#)
 - RCC, [209](#)
- RCC_PLLSource_HSE
 - RCC_PLL_Clock_Source, [246](#)
- RCC_PLLSource_HSI
 - RCC_PLL_Clock_Source, [246](#)
- RCC_Private_Functions, [212](#)
- RCC_RTC_Clock_Source, [250](#)
 - IS_RCC_RTCCLK_SOURCE, [251](#)
 - RCC_RTCCLKSource_HSE_Div10, [251](#)
 - RCC_RTCCLKSource_HSE_Div11, [251](#)
 - RCC_RTCCLKSource_HSE_Div12, [251](#)
 - RCC_RTCCLKSource_HSE_Div13, [251](#)
 - RCC_RTCCLKSource_HSE_Div14, [251](#)
 - RCC_RTCCLKSource_HSE_Div15, [251](#)
 - RCC_RTCCLKSource_HSE_Div16, [251](#)
 - RCC_RTCCLKSource_HSE_Div17, [251](#)
 - RCC_RTCCLKSource_HSE_Div18, [251](#)
 - RCC_RTCCLKSource_HSE_Div19, [251](#)
 - RCC_RTCCLKSource_HSE_Div2, [252](#)

- RCC_RTCCLKSource_HSE_Div20, [252](#)
- RCC_RTCCLKSource_HSE_Div21, [252](#)
- RCC_RTCCLKSource_HSE_Div22, [252](#)
- RCC_RTCCLKSource_HSE_Div23, [252](#)
- RCC_RTCCLKSource_HSE_Div24, [252](#)
- RCC_RTCCLKSource_HSE_Div25, [252](#)
- RCC_RTCCLKSource_HSE_Div26, [252](#)
- RCC_RTCCLKSource_HSE_Div27, [252](#)
- RCC_RTCCLKSource_HSE_Div28, [252](#)
- RCC_RTCCLKSource_HSE_Div29, [252](#)
- RCC_RTCCLKSource_HSE_Div3, [252](#)
- RCC_RTCCLKSource_HSE_Div30, [252](#)
- RCC_RTCCLKSource_HSE_Div31, [252](#)
- RCC_RTCCLKSource_HSE_Div4, [252](#)
- RCC_RTCCLKSource_HSE_Div5, [253](#)
- RCC_RTCCLKSource_HSE_Div6, [253](#)
- RCC_RTCCLKSource_HSE_Div7, [253](#)
- RCC_RTCCLKSource_HSE_Div8, [253](#)
- RCC_RTCCLKSource_HSE_Div9, [253](#)
- RCC_RTCCLKSource_LSE, [253](#)
- RCC_RTCCLKSource_LSI, [253](#)
- RCC_RTCCLKCmd
 - Peripheral clocks configuration functions, [240](#)
 - RCC, [210](#)
- RCC_RTCCLKConfig
 - Peripheral clocks configuration functions, [240](#)
 - RCC, [210](#)
- RCC_RTCCLKSource_HSE_Div10
 - RCC_RTC_Clock_Source, [251](#)
- RCC_RTCCLKSource_HSE_Div11
 - RCC_RTC_Clock_Source, [251](#)
- RCC_RTCCLKSource_HSE_Div12
 - RCC_RTC_Clock_Source, [251](#)
- RCC_RTCCLKSource_HSE_Div13
 - RCC_RTC_Clock_Source, [251](#)
- RCC_RTCCLKSource_HSE_Div14
 - RCC_RTC_Clock_Source, [251](#)
- RCC_RTCCLKSource_HSE_Div15
 - RCC_RTC_Clock_Source, [251](#)
- RCC_RTCCLKSource_HSE_Div16
 - RCC_RTC_Clock_Source, [251](#)
- RCC_RTCCLKSource_HSE_Div17
 - RCC_RTC_Clock_Source, [251](#)
- RCC_RTCCLKSource_HSE_Div18
 - RCC_RTC_Clock_Source, [251](#)
- RCC_RTCCLKSource_HSE_Div19
 - RCC_RTC_Clock_Source, [251](#)
- RCC_RTCCLKSource_HSE_Div2
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div20
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div21
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div22
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div23
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div24
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div25
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div26
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div27
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div28
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div29
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div3
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div30
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div31
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div4
 - RCC_RTC_Clock_Source, [252](#)
- RCC_RTCCLKSource_HSE_Div5
 - RCC_RTC_Clock_Source, [253](#)
- RCC_RTCCLKSource_HSE_Div6
 - RCC_RTC_Clock_Source, [253](#)
- RCC_RTCCLKSource_HSE_Div7
 - RCC_RTC_Clock_Source, [253](#)
- RCC_RTCCLKSource_HSE_Div8
 - RCC_RTC_Clock_Source, [253](#)
- RCC_RTCCLKSource_HSE_Div9
 - RCC_RTC_Clock_Source, [253](#)
- RCC_RTCCLKSource_LSE
 - RCC_RTC_Clock_Source, [253](#)
- RCC_RTCCLKSource_LSI
 - RCC_RTC_Clock_Source, [253](#)
- RCC_SYSCLK_Div1
 - RCC_AHB_Clock_Source, [247](#)
- RCC_SYSCLK_Div128
 - RCC_AHB_Clock_Source, [247](#)
- RCC_SYSCLK_Div16
 - RCC_AHB_Clock_Source, [247](#)
- RCC_SYSCLK_Div2
 - RCC_AHB_Clock_Source, [247](#)
- RCC_SYSCLK_Div256
 - RCC_AHB_Clock_Source, [247](#)
- RCC_SYSCLK_Div4
 - RCC_AHB_Clock_Source, [247](#)
- RCC_SYSCLK_Div512
 - RCC_AHB_Clock_Source, [247](#)
- RCC_SYSCLK_Div64
 - RCC_AHB_Clock_Source, [248](#)
- RCC_SYSCLK_Div8
 - RCC_AHB_Clock_Source, [248](#)
- RCC_SYSCLKConfig
 - RCC, [211](#)
 - System AHB and APB busses clocks configuration functions, [225](#)
- RCC_SYSCLKSource_HSE
 - RCC_System_Clock_Source, [246](#)
- RCC_SYSCLKSource_HSI

- RCC_System_Clock_Source, 246
- RCC_SYSClkSource_PLLCLK
 - RCC_System_Clock_Source, 247
- RCC_System_Clock_Source, 246
 - IS_RCC_SYSClk_SOURCE, 246
 - RCC_SYSClkSource_HSE, 246
 - RCC_SYSClkSource_HSI, 246
 - RCC_SYSClkSource_PLLCLK, 247
- RCC_WaitForHSEStartUp
 - Internal and external clocks, PLL, CSS and MCO configuration functions, 221
- RCC, 211
- Regular Channels Configuration functions, 41
 - ADC_ContinuousModeCmd, 42
 - ADC_DiscModeChannelCountConfig, 42
 - ADC_DiscModeCmd, 43
 - ADC_EOCOnEachRegularChannelCmd, 43
 - ADC_GetConversionValue, 44
 - ADC_GetMultiModeConversionValue, 44
 - ADC_GetSoftwareStartConvStatus, 44
 - ADC_RegularChannelConfig, 45
 - ADC_SoftwareStartConv, 47
- Regular Channels DMA Configuration functions, 47
 - ADC_DMACmd, 48
 - ADC_DMAResquestAfterLastTransferCmd, 48
 - ADC_MultiModeDMAResquestAfterLastTransferCmd, 48
- Reset
 - gpio.h, 345
- Resolution
 - _ADC_InitTypeDef, 312
- Rising
 - gpio.h, 346
- RTCEN_BitNumber
 - RCC, 184
- RTE/Target_1/RTE_Components.h, 422
- RTE/Device/STM32F401RETx/system_stm32f4xx.c, 423
- RTE_Components.h
 - CMSIS_device_header, 422
 - RTE_DEVICE_STARTUP_STM32F4XX, 422
- RTE_DEVICE_STARTUP_STM32F4XX
 - RTE_Components.h, 422
- SamplingTime
 - ADC_ChannelConfTypeDef, 313
- ScanConvMode
 - _ADC_InitTypeDef, 312
- Smartcard mode functions, 288
 - USART_SetGuardTime, 289
 - USART_SmartCardCmd, 289
 - USART_SmartCardNACKCmd, 290
- SMPR1_SMP_SET
 - ADC, 13
- SMPR2_SMP_SET
 - ADC, 13
- Speed
 - GPIO_InitTypeDef, 317
- SQR1_L_RESET
 - ADC, 13
- SQR1_SQ_SET
 - ADC, 13
- SQR2_SQ_SET
 - ADC, 13
- SQR3_SQ_SET
 - ADC, 14
- src/main.c, 424
- State
 - ADC_HandleTypeDef, 314
- stm32f4xx_rcc.c
 - HSE_STARTUP_TIMEOUT, 396
 - HSE_VALUE, 396
 - HSI_VALUE, 396
- STM32F4xx_StdPeriph_Driver, 9
- Stm32f4xx_system, 308
- STM32F4xx_System_Private_Defines, 308
- STM32F4xx_System_Private_FunctionPrototypes, 309
- STM32F4xx_System_Private_Functions, 309
 - SystemCoreClockUpdate, 309
 - SystemInit, 309
- STM32F4xx_System_Private_Includes, 308
 - HSE_VALUE, 308
 - HSI_VALUE, 308
- STM32F4xx_System_Private_Macros, 308
- STM32F4xx_System_Private_TypesDefinitions, 308
- STM32F4xx_System_Private_Variables, 308
 - AHBPrescTable, 308
 - APBPrescTable, 308
 - SystemCoreClock, 308
- STM_MODE_ANALOG
 - adc.h, 332
- STM_PIN
 - adc.h, 332
- STM_PIN_AFNUM
 - adc.h, 332
- STM_PIN_CHANNEL
 - adc.h, 333
- STM_PIN_DATA_EXT
 - adc.h, 333
- STM_PIN_MODE
 - adc.h, 333
- STM_PIN_PUPD
 - adc.h, 333
- STM_PORT
 - adc.h, 333
- SYSClk_Frequency
 - RCC_ClocksTypeDef, 319
- System AHB and APB busses clocks configuration functions, 221
 - RCC_GetClocksFreq, 222
 - RCC_GetSYSClkSource, 223
 - RCC_HCLKConfig, 223
 - RCC_PCLK1Config, 224
 - RCC_PCLK2Config, 225
 - RCC_SYSClkConfig, 225
- SystemCoreClock
 - STM32F4xx_System_Private_Variables, 308

- SystemCoreClockUpdate
 - STM32F4xx_System_Private_Functions, 309
- SystemInit
 - STM32F4xx_System_Private_Functions, 309
- SysTick_Handler
 - timer.c, 416
- Temperature Sensor, Vrefint (Voltage Reference internal), 40
 - ADC_TempSensorVrefintCmd, 40
 - ADC_VBATCmd, 41
- timer.c
 - SysTick_Handler, 416
 - timer_disable, 416
 - timer_enable, 416
 - timer_init, 416
 - timer_period, 416
 - timer_set_callback, 416
- timer.h
 - timer_disable, 417
 - timer_enable, 417
 - timer_init, 417
 - timer_irq_handler, 417
 - timer_set_callback, 417
- timer_disable
 - timer.c, 416
 - timer.h, 417
- timer_enable
 - timer.c, 416
 - timer.h, 417
- timer_init
 - timer.c, 416
 - timer.h, 417
- timer_irq_handler
 - timer.h, 417
- timer_period
 - timer.c, 416
- timer_set_callback
 - timer.c, 416
 - timer.h, 417
- TriggerMode
 - gpio.h, 346
- uart.c
 - uart_enable, 418
 - uart_init, 418
 - uart_print, 419
 - uart_rx, 419
 - uart_set_rx_callback, 419
 - uart_tx, 419
 - USART2_IRQHandler, 419
- uart.h
 - uart_enable, 420
 - uart_init, 420
 - uart_print, 420
 - uart_rx, 420
 - uart_set_rx_callback, 421
 - uart_tx, 421
- uart_enable
 - uart.c, 418
 - uart.h, 420
- uart_init
 - uart.c, 418
 - uart.h, 420
- uart_print
 - uart.c, 419
 - uart.h, 420
- uart_rx
 - uart.c, 419
 - uart.h, 420
- uart_set_rx_callback
 - uart.c, 419
 - uart.h, 421
- uart_tx
 - uart.c, 419
 - uart.h, 421
- UNUSED
 - adc.h, 333
- USART, 265
 - CR1_CLEAR_MASK, 266
 - CR2_CLOCK_CLEAR_MASK, 266
 - CR3_CLEAR_MASK, 267
 - IT_MASK, 267
 - USART_ClearFlag, 267
 - USART_ClearITPendingBit, 267
 - USART_ClockInit, 268
 - USART_ClockStructInit, 268
 - USART_Cmd, 269
 - USART_DeInit, 269
 - USART_DMAMCmd, 269
 - USART_GetFlagStatus, 270
 - USART_GetITStatus, 270
 - USART_HalfDuplexCmd, 271
 - USART_Init, 271
 - USART_IrDACmd, 272
 - USART_IrDAConfig, 272
 - USART_ITConfig, 272
 - USART_LINBreakDetectLengthConfig, 273
 - USART_LINCmd, 273
 - USART_OneBitMethodCmd, 274
 - USART_OverSampling8Cmd, 274
 - USART_ReceiveData, 274
 - USART_ReceiverWakeUpCmd, 275
 - USART_SendBreak, 275
 - USART_SendData, 275
 - USART_SetAddress, 276
 - USART_SetGuardTime, 276
 - USART_SetPrescaler, 276
 - USART_SmartCardCmd, 277
 - USART_SmartCardNACKCmd, 277
 - USART_StructInit, 277
 - USART_WakeUpConfig, 278
- USART2_IRQHandler
 - uart.c, 419
- USART_BaudRate
 - USART_InitTypeDef, 321
- USART_ClearFlag

- Interrupts and flags management functions, [294](#)
- USART, [267](#)
- USART_ClearITPendingBit
 - Interrupts and flags management functions, [294](#)
 - USART, [267](#)
- USART_Clock, [300](#)
 - IS_USART_CLOCK, [301](#)
 - USART_Clock_Disable, [301](#)
 - USART_Clock_Enable, [301](#)
 - USART_ClockInitTypeDef, [320](#)
- USART_Clock_Disable
 - USART_Clock, [301](#)
- USART_Clock_Enable
 - USART_Clock, [301](#)
- USART_Clock_Phase, [301](#)
 - IS_USART_CPHA, [301](#)
 - USART_CPHA_1Edge, [301](#)
 - USART_CPHA_2Edge, [302](#)
- USART_Clock_Polarity, [301](#)
 - IS_USART_CPOL, [301](#)
 - USART_CPOL_High, [301](#)
 - USART_CPOL_Low, [301](#)
- USART_ClockInit
 - Initialization and Configuration functions, [280](#)
 - USART, [268](#)
- USART_ClockInitTypeDef, [320](#)
 - USART_Clock, [320](#)
 - USART_CPHA, [320](#)
 - USART_CPOL, [320](#)
 - USART_LastBit, [320](#)
- USART_ClockStructInit
 - Initialization and Configuration functions, [280](#)
 - USART, [268](#)
- USART_Cmd
 - Initialization and Configuration functions, [280](#)
 - USART, [269](#)
- USART_CPHA
 - USART_ClockInitTypeDef, [320](#)
- USART_CPHA_1Edge
 - USART_Clock_Phase, [301](#)
- USART_CPHA_2Edge
 - USART_Clock_Phase, [302](#)
- USART_CPOL
 - USART_ClockInitTypeDef, [320](#)
- USART_CPOL_High
 - USART_Clock_Polarity, [301](#)
- USART_CPOL_Low
 - USART_Clock_Polarity, [301](#)
- USART_DeInit
 - Initialization and Configuration functions, [281](#)
 - USART, [269](#)
- USART_DMA_Requests, [304](#)
 - IS_USART_DMAREQ, [304](#)
 - USART_DMAREq_Rx, [304](#)
 - USART_DMAREq_Tx, [304](#)
- USART_DMAMCmd
 - DMA transfers management functions, [292](#)
 - USART, [269](#)
- USART_DMAREq_Rx
 - USART_DMA_Requests, [304](#)
- USART_DMAREq_Tx
 - USART_DMA_Requests, [304](#)
- USART_Exported_Constants, [297](#)
 - IS_USART_1236_PERIPH, [297](#)
 - IS_USART_ALL_PERIPH, [298](#)
- USART_FLAG_CTS
 - USART_Flags, [307](#)
- USART_FLAG_FE
 - USART_Flags, [307](#)
- USART_FLAG_IDLE
 - USART_Flags, [307](#)
- USART_FLAG_LBD
 - USART_Flags, [307](#)
- USART_FLAG_NE
 - USART_Flags, [307](#)
- USART_FLAG_ORE
 - USART_Flags, [307](#)
- USART_FLAG_PE
 - USART_Flags, [307](#)
- USART_FLAG_RXNE
 - USART_Flags, [307](#)
- USART_FLAG_TC
 - USART_Flags, [307](#)
- USART_FLAG_TXE
 - USART_Flags, [307](#)
- USART_Flags, [306](#)
 - IS_USART_ADDRESS, [306](#)
 - IS_USART_BAUDRATE, [306](#)
 - IS_USART_CLEAR_FLAG, [306](#)
 - IS_USART_DATA, [306](#)
 - IS_USART_FLAG, [307](#)
 - USART_FLAG_CTS, [307](#)
 - USART_FLAG_FE, [307](#)
 - USART_FLAG_IDLE, [307](#)
 - USART_FLAG_LBD, [307](#)
 - USART_FLAG_NE, [307](#)
 - USART_FLAG_ORE, [307](#)
 - USART_FLAG_PE, [307](#)
 - USART_FLAG_RXNE, [307](#)
 - USART_FLAG_TC, [307](#)
 - USART_FLAG_TXE, [307](#)
- USART_GetFlagStatus
 - Interrupts and flags management functions, [295](#)
 - USART, [270](#)
- USART_GetITStatus
 - Interrupts and flags management functions, [296](#)
 - USART, [270](#)
- USART_HalfDuplexCmd
 - Halfduplex mode function, [288](#)
 - USART, [271](#)
- USART_Hardware_Flow_Control, [300](#)
 - IS_USART_HARDWARE_FLOW_CONTROL, [300](#)
 - USART_HardwareFlowControl_CTS, [300](#)
 - USART_HardwareFlowControl_None, [300](#)
 - USART_HardwareFlowControl_RTS, [300](#)
 - USART_HardwareFlowControl_RTS_CTS, [300](#)

- USART_HardwareFlowControl
 - USART_InitTypeDef, [321](#)
- USART_HardwareFlowControl_CTS
 - USART_Hardware_Flow_Control, [300](#)
- USART_HardwareFlowControl_None
 - USART_Hardware_Flow_Control, [300](#)
- USART_HardwareFlowControl_RTS
 - USART_Hardware_Flow_Control, [300](#)
- USART_HardwareFlowControl_RTS_CTS
 - USART_Hardware_Flow_Control, [300](#)
- USART_Init
 - Initialization and Configuration functions, [281](#)
 - USART, [271](#)
- USART_InitTypeDef, [320](#)
 - USART_BaudRate, [321](#)
 - USART_HardwareFlowControl, [321](#)
 - USART_Mode, [321](#)
 - USART_Parity, [321](#)
 - USART_StopBits, [321](#)
 - USART_WordLength, [321](#)
- USART_Interrupt_definition, [302](#)
 - IS_USART_CLEAR_IT, [303](#)
 - IS_USART_CONFIG_IT, [303](#)
 - IS_USART_GET_IT, [303](#)
 - USART_IT_CTS, [303](#)
 - USART_IT_ERR, [303](#)
 - USART_IT_FE, [303](#)
 - USART_IT_IDLE, [303](#)
 - USART_IT_LBD, [303](#)
 - USART_IT_NE, [303](#)
 - USART_IT_ORE_ER, [303](#)
 - USART_IT_ORE_RX, [303](#)
 - USART_IT_PE, [304](#)
 - USART_IT_RXNE, [304](#)
 - USART_IT_TC, [304](#)
 - USART_IT_TXE, [304](#)
- USART_IrDA_Low_Power, [305](#)
 - IS_USART_IRDA_MODE, [306](#)
 - USART_IrDAMode_LowPower, [306](#)
 - USART_IrDAMode_Normal, [306](#)
- USART_IrDACmd
 - IrDA mode functions, [291](#)
 - USART, [272](#)
- USART_IrDAConfig
 - IrDA mode functions, [291](#)
 - USART, [272](#)
- USART_IrDAMode_LowPower
 - USART_IrDA_Low_Power, [306](#)
- USART_IrDAMode_Normal
 - USART_IrDA_Low_Power, [306](#)
- USART_IT_CTS
 - USART_Interrupt_definition, [303](#)
- USART_IT_ERR
 - USART_Interrupt_definition, [303](#)
- USART_IT_FE
 - USART_Interrupt_definition, [303](#)
- USART_IT_IDLE
 - USART_Interrupt_definition, [303](#)
- USART_IT_LBD
 - USART_Interrupt_definition, [303](#)
- USART_IT_NE
 - USART_Interrupt_definition, [303](#)
- USART_IT_ORE
 - USART_Legacy, [304](#)
- USART_IT_ORE_ER
 - USART_Interrupt_definition, [303](#)
- USART_IT_ORE_RX
 - USART_Interrupt_definition, [303](#)
- USART_IT_PE
 - USART_Interrupt_definition, [304](#)
- USART_IT_RXNE
 - USART_Interrupt_definition, [304](#)
- USART_IT_TC
 - USART_Interrupt_definition, [304](#)
- USART_IT_TXE
 - USART_Interrupt_definition, [304](#)
- USART_ITConfig
 - Interrupts and flags management functions, [296](#)
 - USART, [272](#)
- USART_Last_Bit, [302](#)
 - IS_USART_LASTBIT, [302](#)
 - USART_LastBit_Disable, [302](#)
 - USART_LastBit_Enable, [302](#)
- USART_LastBit
 - USART_ClockInitTypeDef, [320](#)
- USART_LastBit_Disable
 - USART_Last_Bit, [302](#)
- USART_LastBit_Enable
 - USART_Last_Bit, [302](#)
- USART_Legacy, [304](#)
 - USART_IT_ORE, [304](#)
- USART_LIN_Break_Detection_Length, [305](#)
 - IS_USART_LIN_BREAK_DETECT_LENGTH, [305](#)
 - USART_LINBreakDetectLength_10b, [305](#)
 - USART_LINBreakDetectLength_11b, [305](#)
- USART_LINBreakDetectLength_10b
 - USART_LIN_Break_Detection_Length, [305](#)
- USART_LINBreakDetectLength_11b
 - USART_LIN_Break_Detection_Length, [305](#)
- USART_LINBreakDetectLengthConfig
 - LIN mode functions, [286](#)
 - USART, [273](#)
- USART_LINCmd
 - LIN mode functions, [287](#)
 - USART, [273](#)
- USART_Mode, [299](#)
 - IS_USART_MODE, [300](#)
 - USART_InitTypeDef, [321](#)
 - USART_Mode_Rx, [300](#)
 - USART_Mode_Tx, [300](#)
- USART_Mode_Rx
 - USART_Mode, [300](#)
- USART_Mode_Tx
 - USART_Mode, [300](#)
- USART_OneBitMethodCmd
 - Initialization and Configuration functions, [281](#)

- USART, 274
- USART_OverSampling8Cmd
 - Initialization and Configuration functions, 281
 - USART, 274
- USART_Parity, 299
 - IS_USART_PARITY, 299
 - USART_InitTypeDef, 321
 - USART_Parity_Even, 299
 - USART_Parity_No, 299
 - USART_Parity_Odd, 299
- USART_Parity_Even
 - USART_Parity, 299
- USART_Parity_No
 - USART_Parity, 299
- USART_Parity_Odd
 - USART_Parity, 299
- USART_Private_Functions, 278
- USART_ReceiveData
 - Data transfers functions, 283
 - USART, 274
- USART_ReceiverWakeUpCmd
 - MultiProcessor Communication functions, 284
 - USART, 275
- USART_SendBreak
 - LIN mode functions, 287
 - USART, 275
- USART_SendData
 - Data transfers functions, 283
 - USART, 275
- USART_SetAddress
 - MultiProcessor Communication functions, 285
 - USART, 276
- USART_SetGuardTime
 - Smartcard mode functions, 289
 - USART, 276
- USART_SetPrescaler
 - Initialization and Configuration functions, 282
 - USART, 276
- USART_SmartCardCmd
 - Smartcard mode functions, 289
 - USART, 277
- USART_SmartCardNACKCmd
 - Smartcard mode functions, 290
 - USART, 277
- USART_Stop_Bits, 298
 - IS_USART_STOPBITS, 298
 - USART_StopBits_0_5, 299
 - USART_StopBits_1, 299
 - USART_StopBits_1_5, 299
 - USART_StopBits_2, 299
- USART_StopBits
 - USART_InitTypeDef, 321
- USART_StopBits_0_5
 - USART_Stop_Bits, 299
- USART_StopBits_1
 - USART_Stop_Bits, 299
- USART_StopBits_1_5
 - USART_Stop_Bits, 299
- USART_StopBits_2
 - USART_Stop_Bits, 299
- USART_StopBits_2
 - USART_StopBits_2
 - USART_Stop_Bits, 299
- USART_StructInit
 - Initialization and Configuration functions, 282
 - USART, 277
- USART_WakeUp_AddressMark
 - USART_WakeUp_methods, 305
- USART_WakeUp_IdleLine
 - USART_WakeUp_methods, 305
- USART_WakeUp_methods, 305
 - IS_USART_WAKEUP, 305
 - USART_WakeUp_AddressMark, 305
 - USART_WakeUp_IdleLine, 305
- USART_WakeUpConfig
 - MultiProcessor Communication functions, 285
 - USART, 278
- USART_Word_Length, 298
 - IS_USART_WORD_LENGTH, 298
 - USART_WordLength_8b, 298
 - USART_WordLength_9b, 298
- USART_WordLength
 - USART_InitTypeDef, 321
- USART_WordLength_8b
 - USART_Word_Length, 298
- USART_WordLength_9b
 - USART_Word_Length, 298