

AT32F403 Firmware BSP&Pack

---

## Introduction

This application note is written to give a short description of how to use AT32F403 BSP (Board Support Package) and install AT32 pack.

## Contents

<b>1</b>	<b>Overview .....</b>	<b>36</b>
<b>2</b>	<b>How to install Pack .....</b>	<b>37</b>
2.1	IAR Pack installation.....	37
2.2	Keil_v5 Pack installation.....	39
2.3	Keil_v4 Pack installation.....	40
2.4	Segger Pack installation .....	43
<b>3</b>	<b>Flash algorithm file .....</b>	<b>47</b>
3.1	How to use Keil algorithm file .....	47
3.2	How to use IAR algorithm files.....	49
<b>4</b>	<b>BSP introduction.....</b>	<b>53</b>
4.1	Quick start .....	53
4.1.1	Template project.....	53
4.1.2	BSP macro definitions .....	54
4.2	BSP specifications .....	55
4.2.1	List of abbreviations for peripherals .....	55
4.2.2	Naming rules .....	56
4.2.3	Encoding rules.....	57
4.3	BSP structure .....	60
4.3.1	BSP folder structure .....	60
4.3.2	BSP function library structure.....	61
4.3.3	Initialization and configuration for peripherals .....	62
4.3.4	Peripheral functions description format .....	62
<b>5</b>	<b>AT32F403 peripheral library functions .....</b>	<b>63</b>
5.1	Analog-to-digital converter (ADC).....	63
5.1.1	adc_reset function.....	65
5.1.2	adc_enable function .....	65
5.1.3	adc_combine_mode_select function.....	66
5.1.4	adc_base_default_para_init function .....	67

5.1.5	adc_base_config function .....	67
5.1.6	adc_dma_mode_enable function.....	68
5.1.7	adc_interrupt_enable function.....	69
5.1.8	adc_calibration_init function.....	69
5.1.9	adc_calibration_init_status_get function.....	70
5.1.10	adc_calibration_start function .....	70
5.1.11	adc_calibration_status_get function.....	71
5.1.12	adc_voltage_monitor_enable function .....	71
5.1.13	adc_voltage_monitor_threshold_value_set function .....	72
5.1.14	adc_voltage_monitor_single_channel_select function .....	73
5.1.15	adc_ordinary_channel_set function .....	73
5.1.16	adc_preempt_channel_length_set function .....	74
5.1.17	adc_preempt_channel_set function.....	75
5.1.18	adc_ordinary_conversion_trigger_set function.....	75
5.1.19	adc_preempt_conversion_trigger_set function.....	77
5.1.20	adc_preempt_offset_value_set function .....	78
5.1.21	adc_ordinary_part_count_set function.....	79
5.1.22	adc_ordinary_part_mode_enable function .....	79
5.1.23	adc_preempt_part_mode_enable function .....	80
5.1.24	adc_preempt_auto_mode_enable function .....	80
5.1.25	adc_tempsensor_vinrv_enable function .....	81
5.1.26	adc_ordinary_software_trigger_enable function.....	81
5.1.27	adc_ordinary_software_trigger_status_get function.....	82
5.1.28	adc_preempt_software_trigger_enable function.....	82
5.1.29	adc_preempt_software_trigger_status_get function.....	83
5.1.30	adc_ordinary_conversion_data_get function .....	83
5.1.31	adc_combine_ordinary_conversion_data_get function .....	84
5.1.32	adc_preempt_conversion_data_get function .....	84
5.1.33	adc_flag_get function .....	85
5.1.34	adc_interrupt_flag_get function .....	85
5.1.35	adc_flag_clear function .....	86
5.2	Battery powered domain (BPR).....	87
5.2.1	bpr_reset function.....	89
5.2.2	bpr_flag_get function.....	89
5.2.3	bpr_interrupt_flag_get function .....	90

5.2.4	bpr_flag_clear function.....	90
5.2.5	bpr_interrupt_enable function .....	91
5.2.6	bpr_data_read function .....	91
5.2.7	bpr_data_write function.....	92
5.2.8	bpr_rtc_output_select function.....	92
5.2.9	bpr_rtc_clock_calibration_value_set function.....	93
5.2.10	bpr_tamper_pin_enable function .....	93
5.2.11	bpr_tamper_pin_active_level_set function .....	94
5.3	Controller area network (CAN) .....	95
5.3.1	can_reset function.....	97
5.3.2	can_baudrate_default_para_init function.....	97
5.3.3	can_baudrate_set function.....	98
5.3.4	can_default_para_init function .....	99
5.3.5	can_base_init function .....	99
5.3.6	can_filter_default_para_init function .....	101
5.3.7	can_filter_init function .....	101
5.3.8	can_debug_transmission_prohibit function .....	103
5.3.9	can_ttc_mode_enable function .....	103
5.3.10	can_message_transmit function .....	104
5.3.11	can_transmit_status_get function .....	105
5.3.12	can_transmit_cancel function .....	106
5.3.13	can_message_receive function .....	107
5.3.14	can_receive_fifo_release function .....	108
5.3.15	can_receive_message_pending_get function .....	109
5.3.16	can_operating_mode_set function.....	109
5.3.17	can_doze_mode_enter function.....	110
5.3.18	can_doze_mode_exit function .....	111
5.3.19	can_error_type_record_get function .....	111
5.3.20	can_receive_error_counter_get function .....	112
5.3.21	can_transmit_error_counter_get function .....	112
5.3.22	can_interrupt_enable function.....	113
5.3.23	can_flag_get function .....	114
5.3.24	can_interrupt_flag_get function.....	115
5.3.25	can_flag_clear function .....	116
5.4	CRC calculation unit (CRC).....	117

5.4.1	crc_data_reset function.....	118
5.4.2	crc_one_word_calculate function.....	118
5.4.3	crc_block_calculate function .....	119
5.4.4	crc_data_get function.....	119
5.4.5	crc_common_data_set function .....	120
5.4.6	crc_common_date_get function.....	120
5.4.7	crc_init_data_set function .....	121
5.4.8	crc_reverse_input_data_set function.....	121
5.4.9	crc_reverse_output_data_set function.....	122
5.4.10	crc_poly_value_set function.....	122
5.4.11	crc_poly_value_get function.....	123
5.4.12	crc_poly_size_set function .....	123
5.4.13	crc_poly_size_get function.....	124
5.5	Clock and reset management (CRM) .....	125
5.5.1	crm_reset function.....	127
5.5.2	crm_lext_bypass function.....	127
5.5.3	crm_hext_bypass function .....	128
5.5.4	crm_flag_get function.....	128
5.5.5	crm_interrupt_flag_get function .....	129
5.5.6	crm_hext_stable_wait function.....	130
5.5.7	crm_hick_clock_trimming_set function .....	130
5.5.8	crm_hick_clock_calibration_set function .....	131
5.5.9	crm_periph_clock_enable function .....	131
5.5.10	crm_periph_reset function.....	132
5.5.11	crm_periph_sleep_mode_clock_enable function .....	133
5.5.12	crm_clock_source_enable function.....	134
5.5.13	crm_flag_clear function .....	134
5.5.14	crm_rtc_clock_select function.....	135
5.5.15	crm_rtc_clock_enable function .....	136
5.5.16	crm_ahb_div_set function .....	136
5.5.17	crm_apb1_div_set function .....	137
5.5.18	crm_apb2_div_set function .....	137
5.5.19	crm_adc_clock_div_set function .....	138
5.5.20	crm_usb_clock_div_set function .....	138
5.5.21	crm_clock_failure_detection_enable function.....	139

5.5.22	crm_batteryPoweredDomainReset function.....	139
5.5.23	crm_pllConfig function .....	140
5.5.24	crm_sysclkSwitch function.....	141
5.5.25	crm_sysclkSwitchStatusGet function .....	141
5.5.26	crm_clocksFreqGet function .....	142
5.5.27	crm_clockOutSet function.....	143
5.5.28	crm_interruptEnable function .....	143
5.6	Digital-to-analog converter (DAC).....	144
5.6.1	dacReset function.....	145
5.6.2	dacEnable function .....	145
5.6.3	dacOutputBufferEnable function.....	146
5.6.4	dacTriggerEnable function.....	146
5.6.5	dacTriggerSelect function .....	147
5.6.6	dacSoftwareTriggerGenerate function .....	148
5.6.7	dacDualSoftwareTriggerGenerate function.....	148
5.6.8	dacWaveGenerate function .....	149
5.6.9	dacMaskAmplitudeSelect function .....	149
5.6.10	dacDmaEnable function .....	150
5.6.11	dacDataOutputGet function.....	151
5.6.12	dac1DataSet function .....	151
5.6.13	dac2DataSet function .....	152
5.6.14	dacDualDataSet function .....	153
5.7	Debug.....	154
5.7.1	debugDeviceIdGet function .....	154
5.7.2	debugPeriphModeSet function.....	155
5.8	DMA controller.....	156
5.8.1	dmaDefaultParaInit function.....	158
5.8.2	dmaInit function .....	159
5.8.3	dmaReset function.....	161
5.8.4	dmaDataNumberSet function .....	161
5.8.5	dmaDataNumberGet function .....	162
5.8.6	dmaInterruptEnable function .....	162
5.8.7	dmaChannelEnable function .....	163
5.8.8	dmaFlagGet function .....	163
5.8.9	dmaFlagClear function .....	166

5.9	External interrupt/event controller (EXINT) .....	168
5.9.1	exint_reset function .....	169
5.9.2	exint_default_para_init function .....	169
5.9.3	exint_init function.....	170
5.9.4	exint_flag_clear function .....	171
5.9.5	exint_flag_get function .....	171
5.9.6	exint_interrupt_flag_get function.....	172
5.9.7	exint_software_interrupt_event_generate function.....	172
5.9.8	exint_interrupt_enable function.....	173
5.9.9	exint_event_enable function .....	173
5.10	Flash memory controller (FLASH) .....	174
5.10.1	flash_flag_get function .....	176
5.10.2	flash_flag_clear function .....	176
5.10.3	flash_operation_status_get function .....	178
5.10.4	flash_bank1_operation_status_get function .....	178
5.10.5	flash_bank2_operation_status_get function .....	179
5.10.6	flash_spim_operation_status_get function .....	179
5.10.7	flash_operation_wait_for function .....	180
5.10.8	flash_bank1_operation_wait_for function .....	181
5.10.9	flash_bank2_operation_wait_for function .....	181
5.10.10	flash_spim_operation_wait_for function .....	182
5.10.11	flash_unlock function.....	182
5.10.12	flash_bank1_unlock function.....	183
5.10.13	flash_bank2_unlock function.....	183
5.10.14	flash_spim_unlock function.....	184
5.10.15	flash_lock function.....	184
5.10.16	flash_bank1_lock function.....	184
5.10.17	flash_bank2_lock function.....	185
5.10.18	flash_spim_lock function .....	185
5.10.19	flash_sector_erase function .....	186
5.10.20	flash_internal_all_erase function .....	186
5.10.21	flash_bank1_erase function .....	187
5.10.22	flash_bank2_erase function .....	187
5.10.23	flash_spim_all_erase function.....	188
5.10.24	flash_user_system_data_erase function .....	188

5.10.25 flash_word_program function.....	189
5.10.26 flash_halfword_program function.....	189
5.10.27 flash_byte_program function.....	190
5.10.28 flash_user_system_data_program function.....	191
5.10.29 flash_epp_set function .....	191
5.10.30 flash_epp_status_get function .....	192
5.10.31 flash_fap_enable function .....	193
5.10.32 flash_fap_status_get function .....	193
5.10.33 flash_ssb_set function.....	194
5.10.34 flash_ssb_status_get function.....	195
5.10.35 flash_interrupt_enable function.....	195
5.10.36 flash_spim_model_select function .....	196
5.10.37 flash_spim_encryption_range_set function .....	196
5.11 Geneal-purpose and multiplexed I/Os (GPIO/IOMUX) .....	197
5.11.1 gpio_reset function.....	199
5.11.2 gpio_iomux_reset function .....	199
5.11.3 gpio_init function .....	200
5.11.4 gpio_default_para_init function .....	201
5.11.5 gpio_input_data_bit_read function.....	202
5.11.6 gpio_input_data_read function.....	202
5.11.7 gpio_output_data_bit_read function.....	203
5.11.8 gpio_output_data_read function .....	203
5.11.9 gpio_bits_set function .....	204
5.11.10 gpio_bits_reset function .....	204
5.11.11 gpio_bits_write function.....	205
5.11.12 gpio_port_write function .....	205
5.11.13 gpio_pin_wp_config function.....	206
5.11.14 gpio_event_output_config function .....	207
5.11.15 gpio_event_output_enable function .....	208
5.11.16 gpio_pin_remap_config function .....	208
5.11.17 gpio_exint_line_config function.....	209
5.12 I2C interfaces .....	210
5.12.1 i2c_reset function .....	212
5.12.2 i2c_software_reset function .....	212
5.12.3 i2c_init function.....	213

5.12.4 i2c_own_address1_set function.....	213
5.12.5 i2c_own_address2_set function.....	214
5.12.6 i2c_own_address2_enable function.....	214
5.12.7 i2c_smbus_enable function.....	215
5.12.8 i2c_enable function .....	215
5.12.9 i2c_fast_mode_duty_set function .....	216
5.12.10 i2c_clock_stretch_enable function .....	216
5.12.11 i2c_ack_enable function.....	217
5.12.12 i2c_master_receive_ack_set function.....	217
5.12.13 i2c_pec_position_set function.....	218
5.12.14 i2c_general_call_enable function.....	219
5.12.15 i2c_arp_mode_enable function.....	219
5.12.16 i2c_smbus_mode_set function .....	220
5.12.17 i2c_smbus_alert_set function .....	220
5.12.18 i2c_pec_transmit_enable function .....	221
5.12.19 i2c_pec_calculate_enable function .....	221
5.12.20 i2c_pec_value_get function.....	222
5.12.21 i2c_dma_end_transfer_set function.....	222
5.12.22 i2c_dma_enable function .....	223
5.12.23 i2c_interrupt_enable function .....	223
5.12.24 i2c_start_generate function.....	224
5.12.25 i2c_stop_generate function.....	224
5.12.26 i2c_7bit_address_send function .....	225
5.12.27 i2c_data_send function .....	225
5.12.28 i2c_data_receive function .....	226
5.12.29 i2c_flag_get function .....	226
5.12.30 i2c_interrupt_flag_get function .....	227
5.12.31 i2c_flag_clear function .....	228
5.12.32 i2c_config function.....	229
5.12.33 i2c_lowlevel_init function.....	231
5.12.34 i2c_wait_end function.....	232
5.12.35 i2c_wait_flag function.....	233
5.12.36 i2c_master_transmit function .....	234
5.12.37 i2c_master_receive function .....	235
5.12.38 i2c_slave_transmit function.....	235

5.12.39 i2c_slave_receive function .....	236
5.12.40 i2c_master_transmit_int function .....	236
5.12.41 i2c_master_receive_int function .....	237
5.12.42 i2c_slave_transmit_int function .....	238
5.12.43 i2c_slave_receive_int function .....	238
5.12.44 i2c_master_transmit_dma function .....	239
5.12.45 i2c_master_receive_dma function .....	240
5.12.46 i2c_slave_transmit_dma function .....	240
5.12.47 i2c_slave_receive_dma function .....	241
5.12.48 i2c_memory_write function .....	242
5.12.49 i2c_memory_write_int function .....	243
5.12.50 i2c_memory_write_dma function .....	244
5.12.51 i2c_memory_read function .....	245
5.12.52 i2c_memory_read_int function .....	246
5.12.53 i2c_memory_read_dma function .....	247
5.12.54 i2c_evt_irq_handler function .....	248
5.12.55 i2c_err_irq_handler function .....	248
5.12.56 i2c_dma_tx_irq_handler function .....	249
5.12.57 i2c_dma_rx_irq_handler function .....	249
5.13 Nested vectored interrupt controller (NVIC) .....	250
5.13.1 nvic_system_reset function .....	251
5.13.2 nvic_irq_enable function .....	251
5.13.3 nvic_irq_disable function .....	252
5.13.4 nvic_priority_group_config function .....	252
5.13.5 nvic_vector_table_set function .....	253
5.13.6 nvic_lowpower_mode_config function .....	254
5.14 Power controller (PWC) .....	255
5.14.1 pwc_reset function .....	256
5.14.2 pwc_batteryPowered_domain_access function .....	256
5.14.3 pwc_pvm_level_select function .....	257
5.14.4 pwc_power_voltage_monitor_enable function .....	257
5.14.5 pwc_wakeup_pin_enable function .....	258
5.14.6 pwc_flag_clear function .....	258
5.14.7 pwc_flag_get function .....	259
5.14.8 pwc_sleep_mode_enter function .....	259

5.14.9 pwc_deep_sleep_mode_enter function .....	260
5.14.10 pwc_standby_mode_enter function .....	261
5.15 Real-time clock (RTC) .....	262
5.15.1 rtc_counter_set function.....	263
5.15.2 rtc_counter_get function.....	263
5.15.3 rtc_divider_set function .....	264
5.15.4 rtc_divider_get function .....	264
5.15.5 rtc_alarm_set function .....	265
5.15.6 rtc_interrupt_enable function.....	265
5.15.7 rtc_flag_get function.....	266
5.15.8 rtc_interrupt_flag_get .....	266
5.15.9 rtc_flag_clear function .....	267
5.15.10 rtc_wait_config_finish function .....	267
5.15.11 rtc_wait_update_finish function.....	268
5.16 Serial peripheral interface (SPI)/ I <sup>2</sup> S .....	269
5.16.1 spi_i2s_reset function .....	270
5.16.2 spi_default_para_init function .....	270
5.16.3 spi_init function.....	271
5.16.4 spi_crc_next_transmit function .....	273
5.16.5 spi_crc_polynomial_set function .....	273
5.16.6 spi_crc_polynomial_get function.....	274
5.16.7 spi_crc_enable function .....	274
5.16.8 spi_crc_value_get function.....	275
5.16.9 spi_hardware_cs_output_enable function .....	275
5.16.10 spi_software_cs_internal_level_set function .....	276
5.16.11 spi_frame_bit_num_set function .....	277
5.16.12 spi_half_duplex_direction_set function.....	277
5.16.13 spi_enable function .....	278
5.16.14 i2s_default_para_init function .....	278
5.16.15 i2s_init function.....	279
5.16.16 i2s_enable function .....	280
5.16.17 spi_i2s_interrupt_enable function .....	281
5.16.18 spi_i2s_dma_transmitter_enable function .....	282
5.16.19 spi_i2s_dma_receiver_enable function.....	282
5.16.20 spi_i2s_data_transmit function .....	283

5.16.21 spi_i2s_data_receive function.....	283
5.16.22 spi_i2s_flag_get function.....	284
5.16.23 spi_i2s_interrupt_flag_get function .....	285
5.16.24 spi_i2s_flag_clear function.....	286
5.17 SysTick.....	287
5.17.1 systick_clock_source_config function .....	287
5.17.2 SysTick_Config function.....	288
5.18 SDIO interfaces .....	289
5.18.1 sdio_reset function .....	291
5.18.2 sdio_power_set function .....	291
5.18.3 sdio_power_status_get function .....	292
5.18.4 sdio_clock_config function .....	292
5.18.5 sdio_bus_width_config function .....	293
5.18.6 sdio_clock_bypass function .....	293
5.18.7 sdio_power_saving_mode_enable function.....	294
5.18.8 sdio_flow_control_enable function.....	294
5.18.9 sdio_clock_enable function.....	295
5.18.10 sdio_dma_enable function .....	295
5.18.11 sdio_interrupt_enable function.....	296
5.18.12 sdio_flag_get function .....	297
5.18.13 sdio_interrupt_flag_get function.....	298
5.18.14 sdio_flag_clear function .....	299
5.18.15 sdio_command_config function .....	300
5.18.16 sdio_command_state_machine_enable function.....	301
5.18.17 sdio_command_response_get function .....	301
5.18.18 sdio_response_get function .....	302
5.18.19 sdio_data_config function .....	302
5.18.20 sdio_data_state_machine_enable function .....	304
5.18.21 sdio_data_counter_get function.....	304
5.18.22 sdio_data_read function.....	305
5.18.23 sdio_buffer_counter_get function.....	305
5.18.24 sdio_data_write function .....	306
5.18.25 sdio_read_wait_mode_set function .....	306
5.18.26 sdio_read_wait_start function .....	307
5.18.27 sdio_read_wait_stop function .....	307

5.18.28 sdio_io_function_enable function.....	308
5.18.29 sdio_io_suspend_command_set function.....	308
5.19 TMR .....	309
5.19.1 tmr_reset function.....	311
5.19.2 tmr_counter_enable function.....	311
5.19.3 tmr_output_default_para_init function.....	312
5.19.4 tmr_input_default_para_init function.....	312
5.19.5 tmr_brkdt_default_para_init function.....	313
5.19.6 tmr_base_init function .....	314
5.19.7 tmr_clock_source_div_set function.....	314
5.19.8 tmr_cnt_dir_set function.....	315
5.19.9 tmr_repetition_counter_set function.....	315
5.19.10 tmr_counter_value_set function.....	316
5.19.11 tmr_counter_value_get function.....	316
5.19.12 tmr_div_value_set function .....	317
5.19.13 tmr_div_value_get function .....	317
5.19.14 tmr_output_channel_config function.....	318
5.19.15 tmr_output_channel_mode_select function.....	320
5.19.16 tmr_period_value_set function.....	321
5.19.17 tmr_period_value_get function.....	321
5.19.18 tmr_channel_value_set function .....	322
5.19.19 tmr_channel_value_get function .....	322
5.19.20 tmr_period_buffer_enable function .....	323
5.19.21 tmr_output_channel_buffer_enable function .....	324
5.19.22 tmr_output_channel_immediately_set function .....	324
5.19.23 tmr_output_channel_switch_set function.....	325
5.19.24 tmr_one_cycle_mode_enable function .....	326
5.19.25 tmr_32_bit_function_enable function .....	326
5.19.26 tmr_overflow_request_source_set function .....	327
5.19.27 tmr_overflow_event_disable function.....	328
5.19.28 tmr_input_channel_init function .....	328
5.19.29 tmr_channel_enable function .....	330
5.19.30 tmr_input_channel_filter_set function .....	331
5.19.31 tmr_pwm_input_config function .....	331
5.19.32 tmr_channel1_input_select function .....	332

5.19.33 tmr_input_channel_divider_set function .....	333
5.19.34 tmr_primary_mode_select function.....	333
5.19.35 tmr_sub_mode_select function .....	334
5.19.36 tmr_channel_dma_select function .....	335
5.19.37 tmr_hall_select function .....	335
5.19.38 tmr_channel_buffer_enable function.....	336
5.19.39 tmr_trigger_input_select function.....	336
5.19.40 tmr_sub_sync_mode_set function .....	337
5.19.41 tmr_dma_request_enable function .....	337
5.19.42 tmr_interrupt_enable function .....	338
5.19.43 tmr_interrupt_flag_get function .....	339
5.19.44 tmr_flag_get function.....	340
5.19.45 tmr_flag_clear function.....	341
5.19.46 tmr_event_sw_trigger function.....	341
5.19.47 tmr_output_enable function.....	342
5.19.48 tmr_internal_clock_set function .....	342
5.19.49 tmr_output_channel_polarity_set function .....	343
5.19.50 tmr_external_clock_config function.....	343
5.19.51 tmr_external_clock_mode1_config function .....	344
5.19.52 tmr_external_clock_mode2_config function .....	345
5.19.53 tmr_encoder_mode_config function.....	346
5.19.54 tmr_force_output_set function .....	347
5.19.55 tmr_dma_control_config function.....	347
5.19.56 tmr_brkdt_config function.....	348
5.20 Universal synchronous/asynchronous receiver/transmitter (USART) .....	350
5.20.1 usart_reset function.....	351
5.20.2 usart_init function .....	352
5.20.3 usart_parity_selection_config function.....	353
5.20.4 usart_enable function.....	353
5.20.5 usart_transmitter_enable function.....	354
5.20.6 usart_receiver_enable function.....	354
5.20.7 usart_clock_config function.....	355
5.20.8 usart_clock_enable function.....	356
5.20.9 usart_interrupt_enable function .....	356
5.20.10 usart_dma_transmitter_enable function.....	357

5.20.11 usart_dma_receiver_enable function.....	357
5.20.12 usart_wakeup_id_set function .....	358
5.20.13 usart_wakeup_mode_set function .....	358
5.20.14 usart_receiver_mute_enable function.....	359
5.20.15 usart_break_bit_num_set function.....	359
5.20.16 usart_lin_mode_enable function .....	360
5.20.17 usart_data_transmit function.....	360
5.20.18 usart_data_receive function.....	361
5.20.19 usart_break_send function.....	361
5.20.20 usart_smartcard_guard_time_set function .....	362
5.20.21 usart_irda_smartcard_division_set function .....	362
5.20.22 usart_smartcard_mode_enable function .....	363
5.20.23 usart_smartcard_nack_set function.....	363
5.20.24 usart_single_line_halfduplex_select function .....	364
5.20.25 usart_irda_mode_enable function.....	364
5.20.26 usart_irda_low_power_enable function .....	365
5.20.27 usart_hardware_flow_control_set function .....	365
5.20.28 usart_flag_get function.....	366
5.20.29 usart_interrupt_flag_get function .....	367
5.20.30 usart_flag_clear function .....	368
5.21 Watchdog timer (WDT).....	369
5.21.1 wdt_enable function .....	370
5.21.2 wdt_counter_reload function.....	370
5.21.3 wdt_reload_value_set function .....	371
5.21.4 wdt_divider_set function.....	371
5.21.5 wdt_register_write_enable function .....	372
5.21.6 wdt_flag_get function .....	372
5.22 Window watchdog timer (WWDT).....	373
5.22.1 wwdt_reset function.....	374
5.22.2 wwdt_divider_set function .....	374
5.22.3 wwdt_enable function.....	375
5.22.4 wwdt_interrupt_enable function .....	375
5.22.5 wwdt_counter_set function .....	375
5.22.6 wwdt_window_counter_set function .....	376
5.22.7 wwdt_flag_get function.....	376

5.22.8	wwdt_interrupt_flag_get function .....	377
5.22.9	wwdt_flag_clear function.....	377
5.23	External memory controller (XMC) .....	378
5.23.1	xmc_nor_sram_reset function.....	381
5.23.2	xmc_nor_sram_init function .....	381
5.23.3	xmc_nor_sram_timing_config function .....	384
5.23.4	xmc_norsram_default_para_init function .....	386
5.23.5	xmc_norsram_timing_default_para_init function .....	387
5.23.6	xmc_nor_sram_enable function.....	388
5.23.7	xmc_ext_timing_config function.....	389
5.23.8	xmc_nand_reset function.....	390
5.23.9	xmc_nand_init function .....	390
5.23.10	xmc_nand_timing_config function.....	392
5.23.11	xmc_nand_default_para_init function .....	393
5.23.12	xmc_nand_timing_default_para_init function .....	394
5.23.13	xmc_nand_enable function .....	395
5.23.14	xmc_nand_ecc_enable function .....	395
5.23.15	xmc_ecc_get function .....	396
5.23.16	xmc_interrupt_enable function.....	396
5.23.17	xmc_flag_status_get function .....	397
5.23.18	xmc_interrupt_flag_clear function .....	398
5.23.19	xmc_flag_clear function .....	399
5.23.20	xmc_pccard_reset function .....	399
5.23.21	xmc_pccard_init function.....	400
5.23.22	xmc_pccard_timing function.....	401
5.23.23	xmc_pccard_default_para_init function .....	403
5.23.24	xmc_pccard_timing_default_para_init function.....	404
5.23.25	xmc_pccard_enable function .....	405
6	Precautions .....	406
6.1	Device model replacement .....	406
6.1.1	KEIL environment.....	406
6.1.2	IAR environment.....	407
6.2	Failed to identify IC by JLink software in Keil.....	409
6.3	How to change HEXT crystal .....	411

---

7	Revision history.....	412
---	-----------------------	-----

## List of tables

Table 1. Summary of macro definitions .....	54
Table 2. List of abbreviations for peripherals.....	55
Table 3. Summary of BSP function library files .....	61
Table 4. Function format description for peripherals .....	62
Table 5. Summary of ADC registers .....	63
Table 6. Summary of ADC library functions.....	64
Table 7. adc_reset function.....	65
Table 8. adc_enable function.....	65
Table 9. adc_combine_mode_select function .....	66
Table 10. adc_base_default_para_init function .....	67
Table 11. adc_base_config function .....	67
Table 12. adc_dma_mode_enable function.....	68
Table 13. adc_interrupt_enable function .....	69
Table 14. adc_calibration_init function.....	69
Table 15. adc_calibration_init_status_get function.....	70
Table 16. adc_calibration_start function .....	70
Table 17. adc_calibration_status_get function .....	71
Table 18. adc_voltage_monitor_enable function.....	71
Table 19. adc_voltage_monitor_threshold_value_set function .....	72
Table 20. adc_voltage_monitor_single_channel_select function .....	73
Table 21. adc_ordinary_channel_set function .....	73
Table 22. adc_preempt_channel_length_set function.....	74
Table 23. adc_preempt_channel_set function.....	75
Table 24. adc_ordinary_conversion_trigger_set function.....	75
Table 25. adc_preempt_conversion_trigger_set function.....	77
Table 26. adc_preempt_offset_value_set function.....	78
Table 27. adc_ordinary_part_count_set function.....	79
Table 28. adc_ordinary_part_mode_enable function .....	79
Table 29. adc_preempt_part_mode_enable .....	80
Table 30. adc_preempt_auto_mode_enable function .....	80
Table 31. adc_tempersensor_vintrv_enable function.....	81
Table 32. adc_ordinary_software_trigger_enable function.....	81
Table 33. adc_ordinary_software_trigger_status_get function.....	82
Table 34. adc_preempt_software_trigger_enable function .....	82

Table 35. adc_preempt_software_trigger_status_get function .....	83
Table 36. adc_ordinary_conversion_data_get function.....	83
Table 37. adc_combine_ordinary_conversion_data_get function .....	84
Table 38. adc_preempt_conversion_data_get function.....	84
Table 39. adc_flag_get function.....	85
Table 40. adc_interrupt_flag_get function .....	85
Table 41. adc_flag_clear function .....	86
Table 42. Summary of BPR registers .....	87
Table 43. Summary of BP library functions.....	88
Table 44. bpr_reset function .....	89
Table 45. bpr_flag_get function .....	89
Table 46. bpr_interrupt_flag_get function .....	90
Table 47. bpr_flag_clear function.....	90
Table 48. bpr_interrupt_enable function .....	91
Table 49. bpr_data_read function .....	91
Table 50. bpr_data_write function.....	92
Table 51. bpr_rtc_output_select function.....	92
Table 52. bpr_rtc_clock_calibration_value_set function.....	93
Table 53. bpr_tamper_pin_enable function .....	93
Table 54. bpr_tamper_pin_active_level_set function .....	94
Table 55. Summary of ADC registers .....	95
Table 56. Summary of CAN library functions.....	96
Table 57. can_reset function.....	97
Table 58. can_baudrate_default_para_init function .....	97
Table 59. can_baudrate_set function.....	98
Table 60. can_default_para_init function .....	99
Table 61. can_base_init function .....	99
Table 62. can_filter_default_para_init function .....	101
Table 63. can_filter_init function .....	101
Table 64. can_debug_transmission_prohibit function .....	103
Table 65. can_ttc_mode_enable function.....	103
Table 66. can_message_transmit function .....	104
Table 67. can_transmit_status_get function .....	105
Table 68. can_transmit_cancel function .....	106
Table 69. can_message_receive function .....	107

Table 70. can_receive_fifo_release function .....	108
Table 71. can_receive_message_pending_get function .....	109
Table 72. can_operating_mode_set function.....	109
Table 73. can_doze_mode_enter function .....	110
Table 74. can_doze_mode_exit function .....	111
Table 75. can_error_type_record_get function.....	111
Table 76. can_receive_error_counter_get function .....	112
Table 77. can_transmit_error_counter_get function.....	112
Table 78. can_interrupt_enable function .....	113
Table 79. can_flag_get function.....	114
Table 80. can_interrupt_flag_get function .....	115
Table 81. can_flag_clear function .....	116
Table 82. Summary of CRC registers .....	117
Table 83. Summary of CRC library functions .....	117
Table 84. crc_data_reset function.....	118
Table 85. crc_one_word_calculate function .....	118
Table 86. crc_block_calculate function .....	119
Table 87. crc_data_get function.....	119
Table 88. crc_common_data_set function.....	120
Table 89. crc_common_date_get function.....	120
Table 90. crc_init_data_set function .....	121
Table 91. crc_reverse_input_data_set function.....	121
Table 92. crc_reverse_output_data_set function.....	122
Table 93. crc_poly_value_set function.....	122
Table 94. crc_poly_value_get function .....	123
Table 95. crc_poly_size_set function.....	123
Table 96. crc_poly_size_get function.....	124
Table 97. Summary of CRM registers.....	125
Table 98. Summary of CRM library functions .....	125
Table 99. crm_reset function.....	127
Table 100. crm_lext_bypass function .....	127
Table 101. crm_hext_bypass function .....	128
Table 102. crm_flag_get function.....	128
Table 103. crm_interrupt_flag_get function .....	129
Table 104. crm_hext_stable_wait function .....	130

Table 105. crm_hick_clock_trimming_set function .....	130
Table 106. crm_hick_clock_calibration_set function .....	131
Table 107. crm_periph_clock_enable function .....	131
Table 108. crm_periph_reset function .....	132
Table 109. crm_periph_sleep_mode_clock_enable function .....	133
Table 110. crm_clock_source_enable function .....	134
Table 111. crm_flag_clear function .....	134
Table 112. crm_erc_clock_select function .....	135
Table 113. crm_erc_clock_enable function .....	136
Table 114. crm_ahb_div_set function .....	136
Table 115. crm_apb1_div_set function .....	137
Table 116. crm_apb2_div_set function .....	137
Table 117. crm_adc_clock_div_set function .....	138
Table 118. crm_usb_clock_div_set function .....	138
Table 119. crm_clock_failure_detection_enable function .....	139
Table 120. crm_batteryPowered_domain_reset function .....	139
Table 121. crm_pll_config function .....	140
Table 122. crm_sysclk_switch function .....	141
Table 123. crm_sysclk_switch_status_get function .....	141
Table 124. crm_clocks_freq_get function .....	142
Table 125. crm_clock_out_set function .....	143
Table 126. crm_interrupt_enable function .....	143
Table 127. Summary of DAC registers .....	144
Table 128. Summary of DAC library functions .....	144
Table 129. dac_reset function .....	145
Table 130. dac_enable function .....	145
Table 131. dac_output_buffer_enable function .....	146
Table 132. dac_trigger_enable function .....	146
Table 133. dac_trigger_select function .....	147
Table 134. dac_software_trigger_generate function .....	148
Table 135. dac_dual_software_trigger_generate function .....	148
Table 136. dac_wave_generate function .....	149
Table 137. dac_mask_amplitude_select function .....	149
Table 138. dac_dma_enable function .....	150
Table 139. dac_data_output_get function .....	151

Table 140. dac_1_data_set function.....	151
Table 141. dac_2_data_set function.....	152
Table 142. dac_dual_data_set function .....	153
Table 143. Summary of debug registers.....	154
Table 144. Summary of debug library functions .....	154
Table 145. debug_device_id_get function .....	154
Table 146. debug_periph_mode_set function .....	155
Table 147. Summary of DMA registers.....	156
Table 148. Summary of DMA library functions .....	157
Table 149. dma_default_para_init function.....	158
Table 150.dma_init_struct default values .....	158
Table 151. dma_init function.....	159
Table 152. dma_reset function .....	161
Table 153. dma_data_number_set function .....	161
Table 154. dma_data_number_get function .....	162
Table 155. dma_interrupt_enable function .....	162
Table 156. dma_channel_enable function.....	163
Table 157. dma_flag_get function.....	163
Table 158. dma_flag_clear function.....	166
Table 159. Summary of EXINT registers .....	168
Table 160. Summary of EXINT library functions.....	168
Table 161. exint_reset function.....	169
Table 162. exint_default_para_init function.....	169
Table 163. exint_init function .....	170
Table 164. exint_flag_clear function .....	171
Table 165. exint_flag_get function .....	171
Table 166. exint_interrupt_flag_get function.....	172
Table 167. exint_software_interrupt_event_generate function .....	172
Table 168. exint_interrupt_enable function.....	173
Table 169. exint_event_enable function .....	173
Table 170. Summary of FLASH registers .....	174
Table 171. Summary of FLASH library functions.....	175
Table 172. flash_flag_get function.....	176
Table 173. flash_flag_clear function .....	177
Table 174. flash_operation_status_get function .....	178

Table 175. flash_bank1_operation_status_get function .....	178
Table 176. flash_bank2_operation_status_get function .....	179
Table 177. flash_spim_operation_status_get function .....	179
Table 178. flash_operation_wait_for function .....	180
Table 179. flash_bank1_operation_wait_for function .....	181
Table 180. flash_bank2_operation_wait_for function .....	181
Table 181. flash_spim_operation_wait_for function .....	182
Table 182. flash_unlock function .....	182
Table 183. flash_bank1_unlock function.....	183
Table 184. flash_bank2_unlock function.....	183
Table 185. flash_spim_unlock function.....	184
Table 186. flash_lock function.....	184
Table 187. flash_bank1_lock function.....	184
Table 188. flash_bank2_lock function.....	185
Table 189. flash_spim_lock function.....	185
Table 190. flash_sector_erase function .....	186
Table 191. flash_internal_all_erase function .....	186
Table 192. flash_bank1_erase function .....	187
Table 193. flash_bank2_erase function .....	187
Table 194. flash_spim_all_erase function .....	188
Table 195. flash_user_system_data_erase function .....	188
Table 196. flash_word_program function .....	189
Table 197. flash_halfword_program function.....	189
Table 198. flash_byte_program function.....	190
Table 199. flash_user_system_data_program function.....	191
Table 200. flash_epp_set function .....	191
Table 201. flash_epp_status_get function .....	192
Table 202. flash_fap_enable function .....	193
Table 203. flash_fap_status_get function .....	193
Table 204. flash_ssb_set function .....	194
Table 205. flash_ssb_status_get function .....	195
Table 206. flash_interrupt_enable function.....	195
Table 207. flash_spim_model_select function.....	196
Table 208. flash_spim_encryption_range_set function .....	196
Table 209. Summary of GPIO registers.....	197

Table 210. Summary of IOMUX registers.....	197
Table 211. GPIO and IOMUX library functions .....	198
Table 212. gpio_reset function.....	199
Table 213. gpio_iomux_reset function .....	199
Table 214. gpio_init function .....	200
Table 215. gpio_default_para_init function.....	201
Table 216. gpio_init_struct default values .....	202
Table 217. gpio_input_data_bit_read function.....	202
Table 218. gpio_input_data_read function .....	202
Table 219. gpio_output_data_bit_read function .....	203
Table 220. gpio_output_data_read function .....	203
Table 221. gpio_bits_set function .....	204
Table 222. gpio_bits_reset function.....	204
Table 227. gpio_bits_write function .....	205
Table 228. gpio_port_write function.....	205
Table 225. gpio_pin_wp_config function .....	206
Table 226. gpio_event_output_config function .....	207
Table 227. gpio_event_output_enable function.....	208
Table 228. gpio_pin_remap_config function .....	208
Table 229. gpio_exint_line_config function.....	209
Table 230. Summary of I2C registera.....	210
Table 231. Summary of I2C library functions.....	210
Table 232. I2C application-layer library functions.....	211
Table 233. i2c_reset function.....	212
Table 234. i2c_software_reset function .....	212
Table 235. i2c_init function .....	213
Table 236. i2c_own_address1_set function .....	213
Table 237. i2c_own_address2_set function .....	214
Table 238. i2c_own_address2_enable function .....	214
Table 239. i2c_smbus_enable function .....	215
Table 240. i2c_enable function .....	215
Table 241. i2c_fast_mode_duty_set function .....	216
Table 242. i2c_clock_stretch_enable function.....	216
Table 243. i2c_ack_enable function .....	217
Table 244. i2c_master_receive_ack_set function.....	217

Table 245. i2c_pec_position_set function.....	218
Table 246. i2c_general_call_enable function .....	219
Table 247. i2c_arp_mode_enable function.....	219
Table 248. i2c_smbus_mode_set function .....	220
Table 249. i2c_smbus_alert_set function .....	220
Table 250. i2c_pec_transmit_enable function .....	221
Table 251. i2c_pec_calculate_enable function.....	221
Table 252. i2c_pec_value_get function .....	222
Table 253. i2c_dma_end_transfer_set function.....	222
Table 254. i2c_dma_enable function .....	223
Table 255. i2c_interrupt_enable function.....	223
Table 256. i2c_start_generate function.....	224
Table 257. i2c_stop_generate function.....	224
Table 258. i2c_7bit_address_send function .....	225
Table 259. i2c_data_send function .....	225
Table 260. i2c_data_receive function .....	226
Table 261. i2c_flag_get function .....	226
Table 262. i2c_interrupt_flag_get function.....	227
Table 263. i2c_flag_clear function .....	228
Table 264. i2c_config function .....	229
Table 265. i2c_lowlevel_init function .....	231
Table 266. i2c_wait_end function .....	232
Table 267. i2c_wait_flag function.....	233
Table 268. i2c_master_transmit function.....	234
Table 269. i2c_master_receive function .....	235
Table 270. i2c_slave_transmit function.....	235
Table 271. i2c_slave_receive function.....	236
Table 272. i2c_master_transmit_int function .....	236
Table 273. i2c_master_receive_int function .....	237
Table 274. 2c_slave_transmit_int function .....	238
Table 275. i2c_slave_receive_int function .....	238
Table 276. i2c_master_transmit_dma function .....	239
Table 277. i2c_master_receive_dma function .....	240
Table 278. i2c_slave_transmit_dma function .....	240
Table 279. i2c_slave_receive_dma function.....	241

Table 280. i2c_memory_write function .....	242
Table 281. i2c_memory_write_int function .....	243
Table 282. i2c_memory_write_dma function .....	244
Table 283. i2c_memory_read function.....	245
Table 284. i2c_memory_read_int function.....	246
Table 285. i2c_memory_read_dma function .....	247
Table 286. i2c_evt_irq_handler function.....	248
Table 287. i2c_err_irq_handler function .....	248
Table 288. i2c_dma_tx_irq_handler function.....	249
Table 289. i2c_dma_rx_irq_handler function.....	249
Table 290. Summary of NVIC registers .....	250
Table 291. Summary of NVIC library functions.....	250
Table 292. nvic_system_reset function.....	251
Table 293. nvic_irq_enable function .....	251
Table 294. nvic_irq_disable function.....	252
Table 295. nvic_priority_group_config function .....	252
Table 296. nvic_vector_table_set function .....	253
Table 297. nvic_lowpower_mode_config function .....	254
Table 298. Summary of PWC registers .....	255
Table 299. Summary of PWC library functions .....	255
Table 300. pwc_reset function .....	256
Table 301. pwc_batteryPowered_domain_access function.....	256
Table 302. pwc_pvm_level_select function .....	257
Table 303. pwc_power_voltage_monitor_enable function .....	257
Table 304. pwc_wakeup_pin_enable function.....	258
Table 305. pwc_flag_clear function .....	258
Table 306. pwc_flag_get function .....	259
Table 307. pwc_sleep_mode_enter function .....	259
Table 308. pwc_deep_sleep_mode_enter function .....	260
Table 309. pwc_standby_mode_enter function .....	261
Table 310. Summary of RTC registers .....	262
Table 311. Summary of RTC library functions .....	262
Table 312. rtc_counter_set function.....	263
Table 313. rtc_counter_get function .....	263
Table 314. rtc_divider_set function .....	264

Table 315. rtc_divider_get function.....	264
Table 316. rtc_alarm_set function.....	265
Table 317. rtc_interrupt_enable function .....	265
Table 318. rtc_flag_get function.....	266
Table 319. rtc_interrupt_flag_get function .....	266
Table 320. rtc_flag_clear function.....	267
Table 321. rtc_wait_config_finish function.....	267
Table 322. rtc_wait_update_finish function .....	268
Table 323. Summary of SPI registers .....	269
Table 324. Summary of SPI library functions .....	269
Table 325. spi_i2s_reset function .....	270
Table 326. spi_default_para_init function .....	270
Table 327. spi_init function .....	271
Table 328. spi_crc_next_transmit function .....	273
Table 329. spi_crc_polynomial_set function.....	273
Table 330. spi_crc_polynomial_get function.....	274
Table 331. spi_crc_enable function .....	274
Table 332. spi_crc_value_get function .....	275
Table 333. spi_hardware_cs_output_enable function .....	275
Table 334. spi_software_cs_internal_level_set function .....	276
Table 335. spi_frame_bit_num_set function .....	277
Table 336. spi_half_duplex_direction_set function.....	277
Table 337. spi_enable function .....	278
Table 338. i2s_default_para_init function .....	278
Table 339. i2s_init function .....	279
Table 340. i2s_enable function .....	280
Table 341. spi_i2s_interrupt_enable function .....	281
Table 342. spi_i2s_dma_transmitter_enable function .....	282
Table 343. spi_i2s_dma_receiver_enable function .....	282
Table 344. spi_i2s_data_transmit function .....	283
Table 345. spi_i2s_data_receive function .....	283
Table 346. spi_i2s_flag_get function .....	284
Table 347. spi_i2s_interrupt_flag_get function .....	285
Table 348. spi_i2s_flag_clear function.....	286
Table 349. Summary of SysTick registers .....	287

Table 350. Summary of SysTick library functions.....	287
Table 351. systick_clock_source_config function.....	287
Table 352. SysTick_Config function.....	288
Table 353. SDIO register description.....	289
Table 354. SDIO library functions .....	290
Table 355. sdio_reset function.....	291
Table 356. sdio_power_set function .....	291
Table 357. sdio_power_status_get function .....	292
Table 358. sdio_clock_config function .....	292
Table 359. sdio_bus_width_config function.....	293
Table 360. sdio_clock_bypass function .....	293
Table 361. sdio_power_saving_mode_enable function .....	294
Table 362. sdio_flow_control_enable function.....	294
Table 363. sdio_clock_enable function.....	295
Table 364. sdio_dma_enable function .....	295
Table 365. sdio_interrupt_enable function.....	296
Table 366. sdio_flag_get function .....	297
Table 367. sdio_interrupt_flag_get function.....	298
Table 368. sdio_flag_clear function .....	299
Table 369. sdio_command_config function .....	300
Table 370. sdio_command_state_machine_enable function .....	301
Table 371. sdio_command_response_get function .....	301
Table 372. sdio_response_get function .....	302
Table 373. sdio_data_config function .....	302
Table 374. sdio_data_state_machine_enable function .....	304
Table 375. sdio_data_counter_get function.....	304
Table 376. sdio_data_read function.....	305
Table 377. sdio_buffer_counter_get function .....	305
Table 378. sdio_data_write function .....	306
Table 379. sdio_read_wait_mode_set function .....	306
Table 380. sdio_read_wait_start function .....	307
Table 381. sdio_read_wait_stop function .....	307
Table 382. sdio_io_function_enable function .....	308
Table 383. sdio_io_suspend_command_set function .....	308
Table 384. Summary of TMR registers .....	309

Table 385. Summary of TMR library functions .....	310
Table 386. tmr_reset function .....	311
Table 387. tmr_counter_enable function .....	311
Table 388. tmr_output_default_para_init function .....	312
Table 389. tmr_output_struct default values.....	312
Table 390. tmr_input_default_para_init function.....	312
Table 391. tmr_input_struct default values.....	313
Table 392. tmr_brkdt_default_para_init function .....	313
Table 393. tmr_brkdt_struct default values.....	313
Table 394. tmr_base_init function.....	314
Table 395. tmr_clock_source_div_set function.....	314
Table 396. tmr_cnt_dir_set function.....	315
Table 397. tmr_repetition_counter_set function .....	315
Table 398. tmr_counter_value_set function.....	316
Table 399. tmr_counter_value_get function .....	316
Table 400. tmr_div_value_set function .....	317
Table 401. tmr_div_value_get function .....	317
Table 402. tmr_output_channel_config function.....	318
Table 403. tmr_output_channel_mode_select function.....	320
Table 404. tmr_period_value_set function.....	321
Table 405. tmr_period_value_get function .....	321
Table 406. tmr_channel_value_set function .....	322
Table 407. tmr_channel_value_get function .....	322
Table 408. tmr_period_buffer_enable function .....	323
Table 409. tmr_output_channel_buffer_enable function .....	324
Table 410. tmr_output_channel_immediately_set function .....	324
Table 411. tmr_output_channel_switch_set function.....	325
Table 412. tmr_one_cycle_mode_enable function.....	326
Table 413. tmr_32_bit_function_enable function.....	326
Table 414. tmr_overflow_request_source_set function .....	327
Table 415. tmr_overflow_event_disable function .....	328
Table 416. tmr_input_channel_init function .....	328
Table 417. tmr_channel_enable function.....	330
Table 418. tmr_input_channel_filter_set function .....	331
Table 419. tmr_pwm_input_config function .....	331

Table 420. tmr_channel1_input_select function .....	332
Table 421. tmr_input_channel_divider_set function .....	333
Table 422. tmr_primary_mode_select function.....	333
Table 423. tmr_sub_mode_select function.....	334
Table 424. tmr_channel_dma_select function .....	335
Table 425. tmr_hall_select function .....	335
Table 426. tmr_channel_buffer_enable function .....	336
Table 427. tmr_trigger_input_select function.....	336
Table 428. tmr_sub_sync_mode_set function .....	337
Table 429. tmr_dma_request_enable function .....	337
Table 430. tmr_interrupt_enable function .....	338
Table 431. tmr_interrupt_flag_get function .....	339
Table 432. tmr_flag_get function .....	340
Table 433. tmr_flag_clear function.....	341
Table 434. tmr_event_sw_trigger function.....	341
Table 435. tmr_output_enable function .....	342
Table 436. tmr_internal_clock_set function .....	342
Table 437. tmr_output_channel_polarity_set function.....	343
Table 438. tmr_external_clock_config function .....	343
Table 439. tmr_external_clock_mode1_config function .....	344
Table 440. tmr_external_clock_mode2_config function .....	345
Table 441. tmr_encoder_mode_config function .....	346
Table 442. tmr_force_output_set function .....	347
Table 443. tmr_dma_control_config function.....	347
Table 444. tmr_brkdt_config function.....	348
Table 445. Summary of USART registers.....	350
Table 446. Summary of USART library functions .....	350
Table 447. usart_reset function .....	351
Table 448. usart_init function .....	352
Table 449. usart_parity_selection_config function .....	353
Table 450. usart_enable function.....	353
Table 451. usart_transmitter_enable function .....	354
Table 452. usart_receiver_enable function.....	354
Table 453. usart_clock_config function.....	355
Table 454. usart_clock_enable function .....	356

Table 455. usart_interrupt_enable function .....	356
Table 456. usart_dma_transmitter_enable function .....	357
Table 457. usart_dma_receiver_enable function.....	357
Table 458. usart_wakeup_id_set function .....	358
Table 459. usart_wakeup_mode_set function.....	358
Table 460. usart_receiver_mute_enable function.....	359
Table 461. usart_break_bit_num_set function.....	359
Table 462. usart_lin_mode_enable function.....	360
Table 463. usart_data_transmit function.....	360
Table 464. usart_data_receive function.....	361
Table 465. usart_break_send function.....	361
Table 466. usart_smartcard_guard_time_set function .....	362
Table 467. usart_irda_smartcard_division_set function .....	362
Table 468. usart_smartcard_mode_enable function .....	363
Table 469. usart_smartcard_nack_set function.....	363
Table 470. usart_single_line_halfduplex_select function .....	364
Table 471. usart_irda_mode_enable function .....	364
Table 472. usart_irda_low_power_enable function .....	365
Table 473. usart_hardware_flow_control_set function .....	365
Table 474. usart_flag_get function.....	366
Table 475. usart_interrupt_flag_get function .....	367
Table 476. usart_flag_clear function.....	368
Table 477. Summary of WDT registers.....	369
Table 478. Summary of WDT library functions .....	369
Table 479. wdt_enable function .....	370
Table 480. wdt_counter_reload function.....	370
Table 481. wdt_reload_value_set function .....	371
Table 482. wdt_divider_set function .....	371
Table 483. wdt_register_write_enable function .....	372
Table 484. wdt_flag_get function .....	372
Table 485. Summary of WWDT registers .....	373
Table 486. Summary of WWDT library functions.....	373
Table 487. wwdt_reset function .....	374
Table 488. wwdt_divider_set function.....	374
Table 489. wwdt_enable function .....	375

Table 490. wwdt_interrupt_enable function .....	375
Table 491. wwdt_counter_set function .....	375
Table 492. wwdt_window_counter_set function .....	376
Table 493. wwdt_flag_get function .....	376
Table 494. wwdt_interrupt_flag_clear function .....	377
Table 495. wwdt_flag_clear function.....	377
Table 496. Summary of XMC registers.....	379
Table 497. Summary of XMC library functions .....	380
Table 498. xmc_nor_sram_reset function .....	381
Table 499. xmc_nor_sram_init function.....	381
Table 500. scfg_mem_map_get function.....	384
Table 501. xmc_norsram_default_para_init function.....	386
Table 502. xmc_nor_sram_init_struct default values .....	387
Table 503. xmc_norsram_timing_default_para_init function .....	387
Table 504. xmc_rw_timing_struct and xmc_w_timing_struct .....	388
Table 505. xmc_nor_sram_enable function.....	388
Table 506. xmc_ext_timing_configfunction.....	389
Table 507. xmc_nand_reset function.....	390
Table 508. xmc_nand_init function .....	390
Table 509. xmc_nand_timing_config function .....	392
Table 510. xmc_nand_default_para_init.....	393
Table 511. xmc_nand_init_struct .....	394
Table 512. xmc_nand_timing_default_para_init function .....	394
Table 513. xmc_common_spacetiming_struct and xmc_attribute_spacetiming_struct .....	394
Table 514. xmc_nand_enable function.....	395
Table 515. xmc_nand_ecc_enable function .....	395
Table 516. xmc_ecc_get function .....	396
Table 517. xmc_interrupt_enable function.....	396
Table 518. xmc_flag_status_get function .....	397
Table 519. xmc_interrupt_flag_clear function .....	398
Table 520. xmc_flag_clear function .....	399
Table 521. xmc_pccard_reset function.....	399
Table 522. xmc_pccard_init function .....	400
Table 523. xmc_pccard_timing function .....	401
Table 524. xmc_pccard_default_para_init function .....	403

Table 525. xmc_pccard_init_struct default value.....	403
Table 526. xmc_pccard_timing_default_para_init function .....	404
Table 527. xmc_common_spacetiming_struct and xmc_attribute_spacetiming_struct.....	404
Table 528. xmc_pccard_enable function .....	405
Table 529. Clock configuration guideline .....	411
Table 530. Document revision history.....	412

## List of figures

Figure 1. Pack kit .....	37
Figure 2. IAR Pack installation window .....	37
Figure 3. IAR Pack installation window .....	38
Figure 4. View IAR Pack installation status .....	39
Figure 5. View Keil_v5 Pack installation status .....	40
Figure 6. Keil_v4 Pack installation status .....	41
Figure 7. Keil_v4 Pack installation process .....	41
Figure 8. Keil_v4 Pack installation complete .....	42
Figure 9. View Keil_v4 Pack installation status .....	43
Figure 10. Segger pack installation window .....	44
Figure 11. Segger pack installation process .....	44
Figure 12. Open J-Flash .....	45
Figure 13. Create a new project using J-Flash .....	45
Figure 14. View Device information .....	46
Figure 15. Keil algorithm file settings .....	47
Figure 16. Keil algorithm file configuration .....	48
Figure 17. Select algorithm files using Keil .....	48
Figure 18. Add algorithm files using Keil .....	49
Figure 19. IAR project name .....	50
Figure 20. IAR algorithm file configuration .....	50
Figure 21. IAR Flash Loader Overview .....	51
Figure 22. IAR Flash Loader configuration .....	51
Figure 23. IAR Flash Loader configuration success .....	52
Figure 24. Template content .....	53
Figure 25. Keil_v5 template project example .....	53
Figure 26. Peripheral enable macro definitions .....	54
Figure 27. BSP folder structure .....	60
Figure 28. BSP function library structure .....	61
Figure 29. Change device part number in Keil .....	406
Figure 30. Change macro definition in Keil .....	407
Figure 31. Change device part number in IAR .....	408
Figure 32. Change macro definition in IAR .....	409
Figure 33. Error warning 1 .....	409
Figure 34. Error warning 2 .....	409

---

Figure 35. Error warning 3 .....	410
Figure 36. JLinkLog and JLinkSettings.....	410
Figure 37. Unspecified Cortex-M4 .....	410
Figure 38. AT32_New_Clock_Configuration window .....	411

## 1 Overview

In order to allow users to make efficient use of Artery MCU, we provide a complete set of BSP & Pack tools to speed up development. They include peripheral driver library, core-related documents and application cases as well as Pack documents supporting a variety of development environments such as Keil\_v5, Keil\_v4, IAR\_v6, IAR\_v7 and IAR\_v8. The BSP and Pack are available on Artery official website.

This application note is written to present how to use BSP and Pack.

## 2 How to install Pack

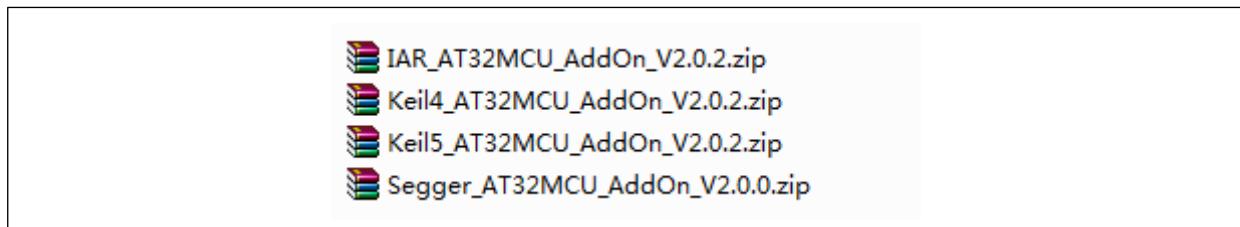
Artery Pack supports various development environment such as Keil\_v5, Keil\_v4, IAR\_v6, IAR\_v7 and IAR\_v8.

Double click on the corresponding Pack to finish installation.

**Note:** This section takes AT32F403A as an example, and other AT32 MCUs have similar Pack installation methods.

The installation package is shown in Figure 1 (the specific version information is subject to the actual conditions).

Figure 1. Pack kit

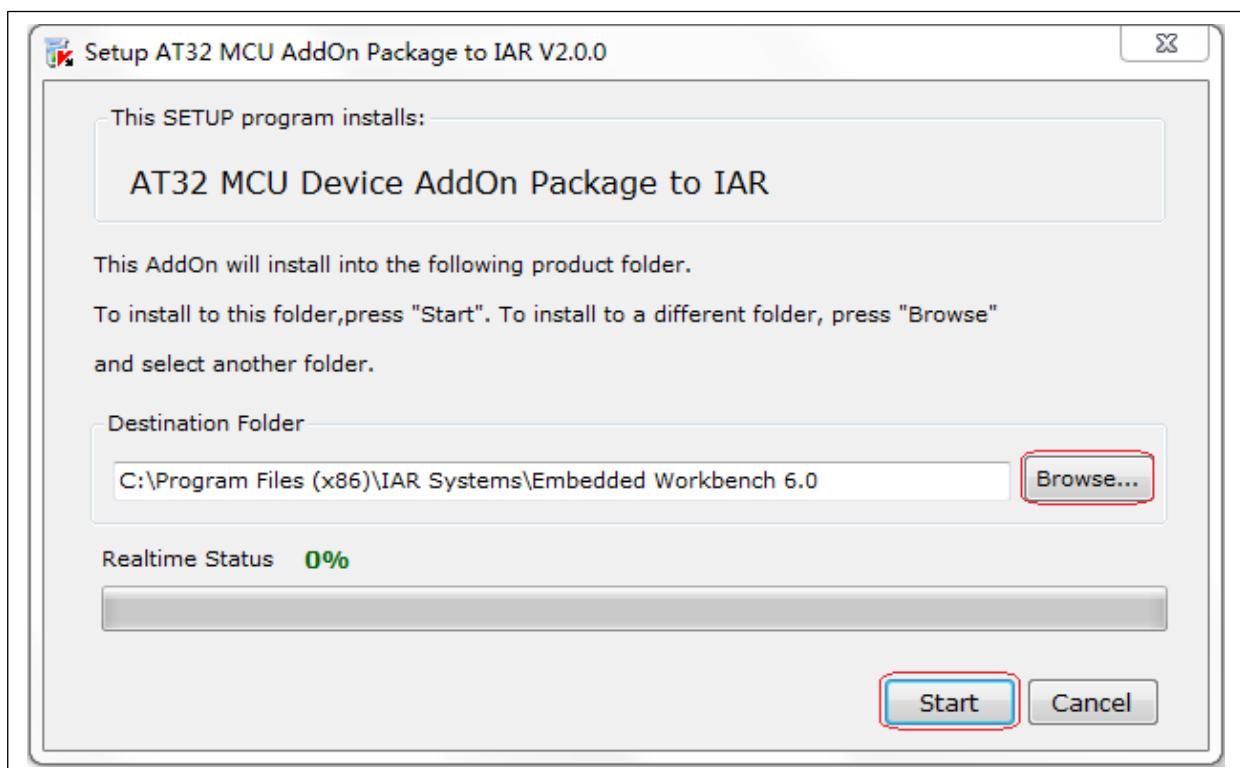


### 2.1 IAR Pack installation

**IAR\_AT32MCU\_AddOn.zip:** This is a zip file supporting IAR\_V6, IAR\_V7 and IAR\_V8. Follow the steps below to install:

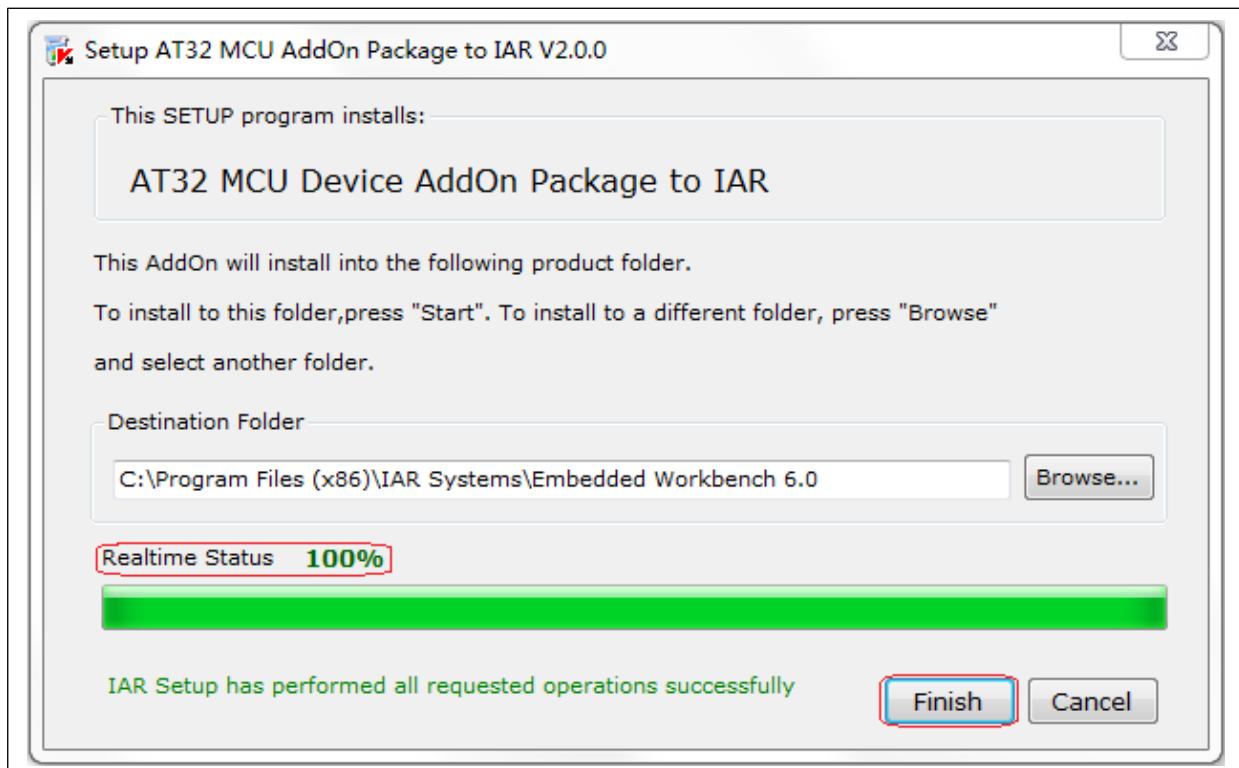
- ① Unzip *IAR\_AT32MCU\_AddOn.zip*
- ② Double click on *IAR\_AT32MCU\_AddOn.exe*, and a dialogue box is displayed below (the specific version information is subject to the actual conditions).

Figure 2. IAR Pack installation window



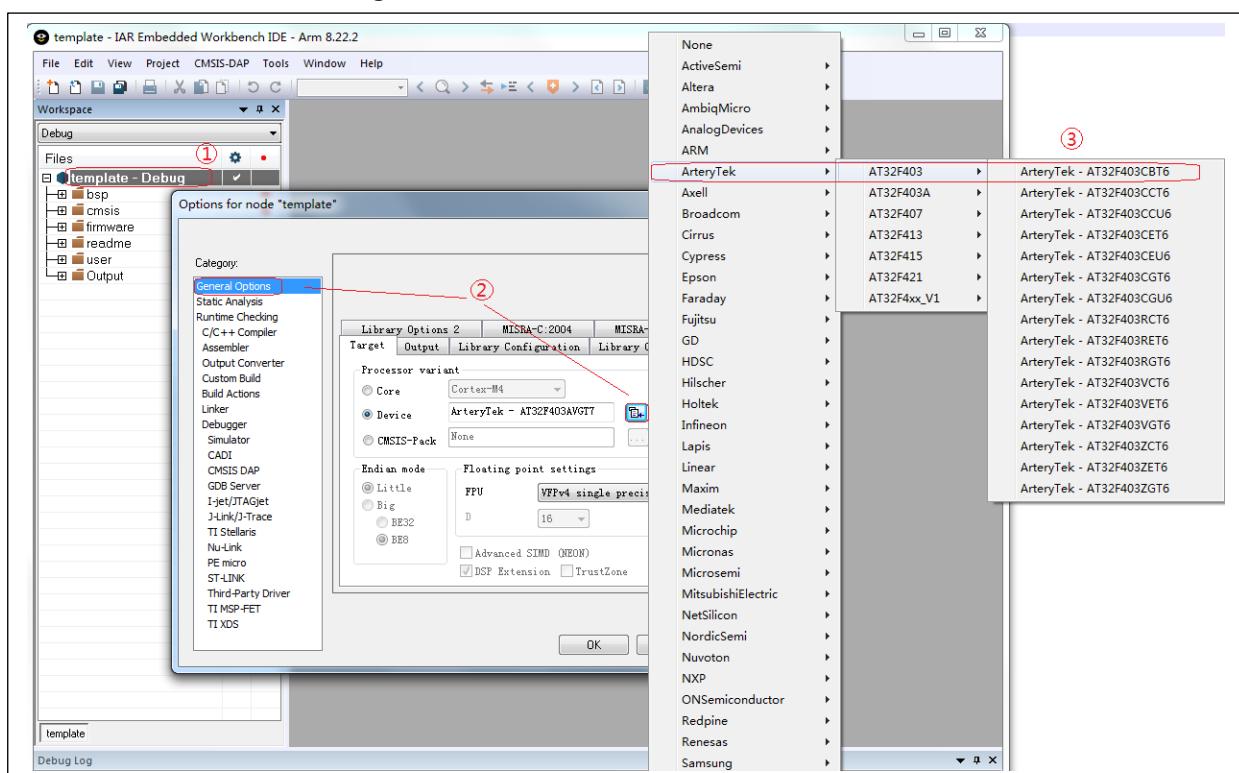
**Note:** If the installation path of IAR does not match the Destination Folder, click on "Browse" to select a correct path, then click on "Start", as shown below.

Figure 3. IAR Pack installation window



- ③ Click on “Finish”
- ④ To check whether the IAR Pack is installed successfully or not, open an IAR project and follow the steps below:
  - Right click on a project name, and select “Options...”
  - Select “General Options”, and click on the check box
  - Click on “ArteryTek” and view AT32 MCU-related information.

Figure 4. View IAR Pack installation status

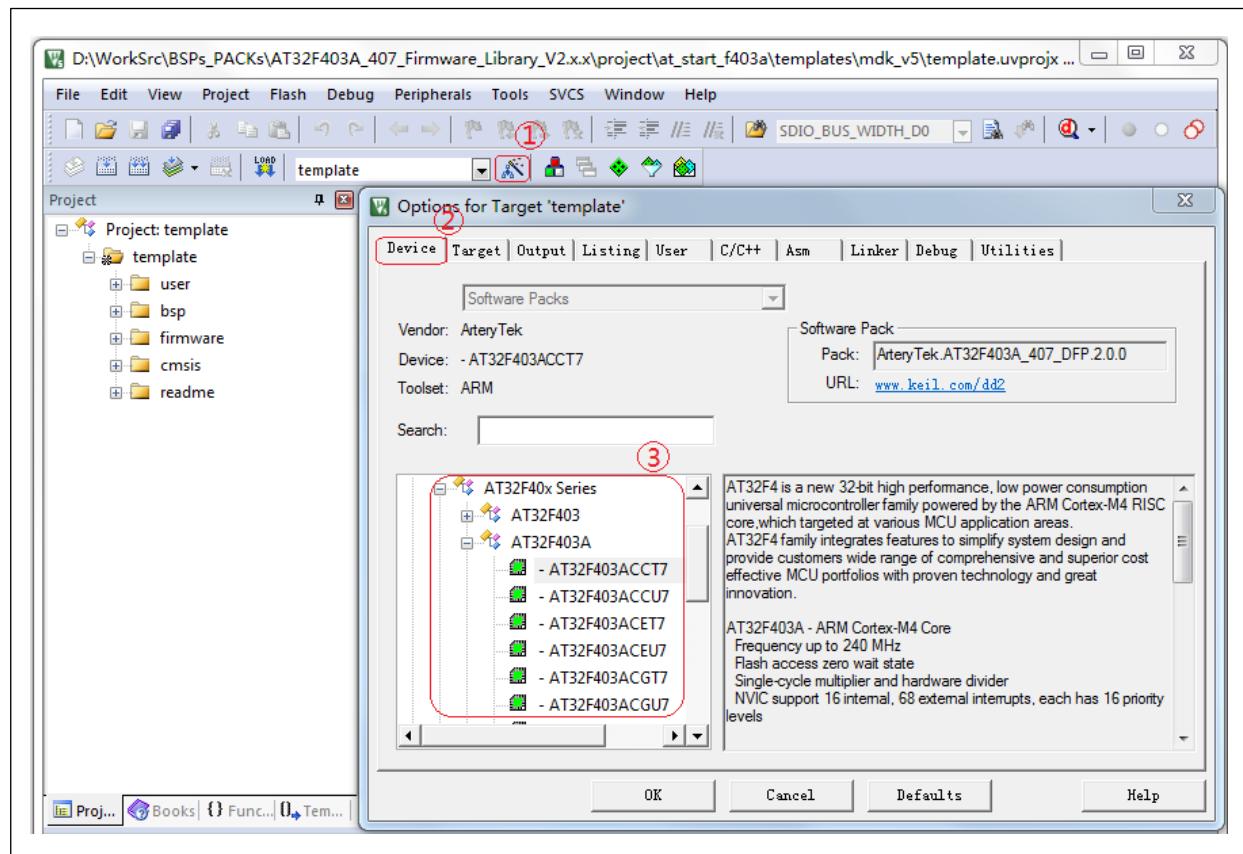


## 2.2 Keil\_v5 Pack installation

**Keil5\_AT32MCU\_AddOn.zip:** This is a zip file supporting Keil\_v5. Follow the steps below to install:

- ① Unzip *Keil5\_AT32MCU\_AddOn.zip*. This zip file includes all Keil5 packs supported, all of which are standard Keil\_v5 DFP installation files.
- ② Select the desired Pack, and double click on *ArteryTek.AT32xxxx\_DFP.2.x.x.pack* to get one-stop installation.
- ⑤ To check whether the Keil\_v5 Pack is installed successfully or not, follow the steps below:
  - Click on wand
  - Select “Device”
  - View AT32 MCU-related information.

Figure 5. View Keil\_v5 Pack installation status

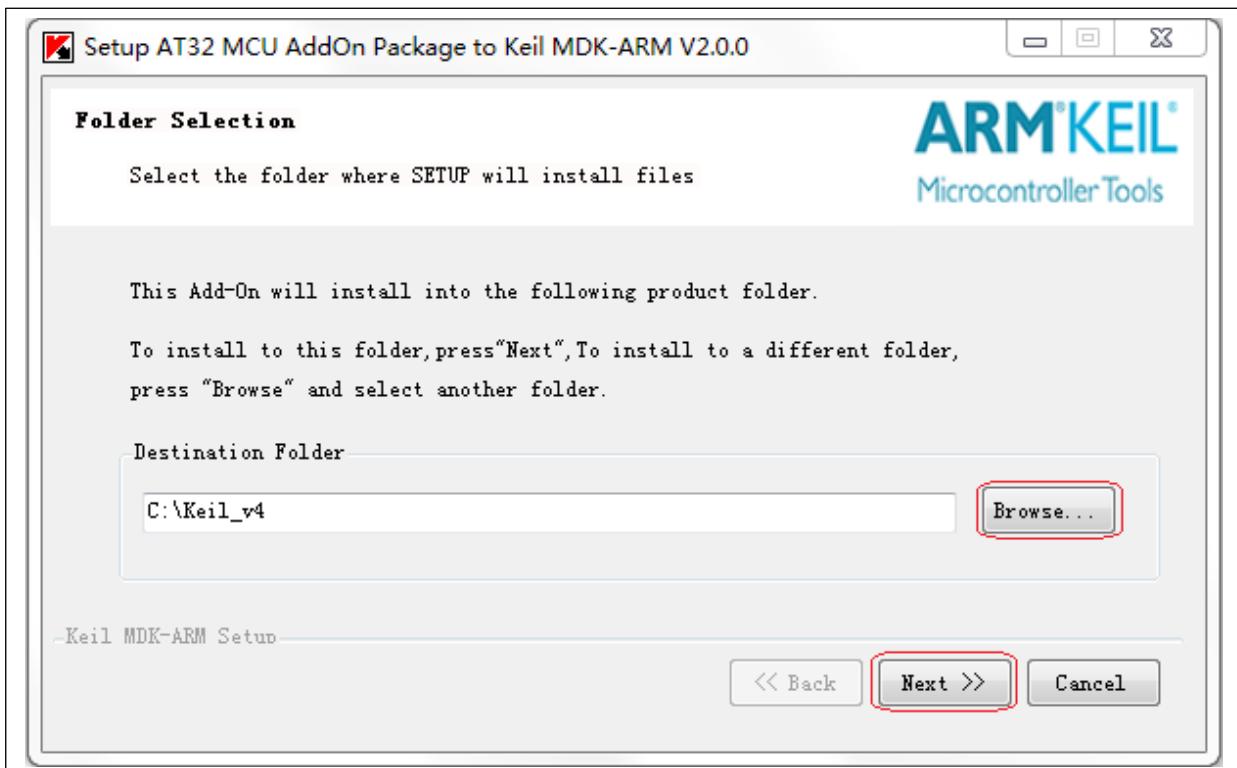


## 2.3 Keil\_v4 Pack installation

**Keil4\_AT32MCU\_AddOn.zip:** This is a zip file supporting Keil\_v4, follow the steps below to finish installation:

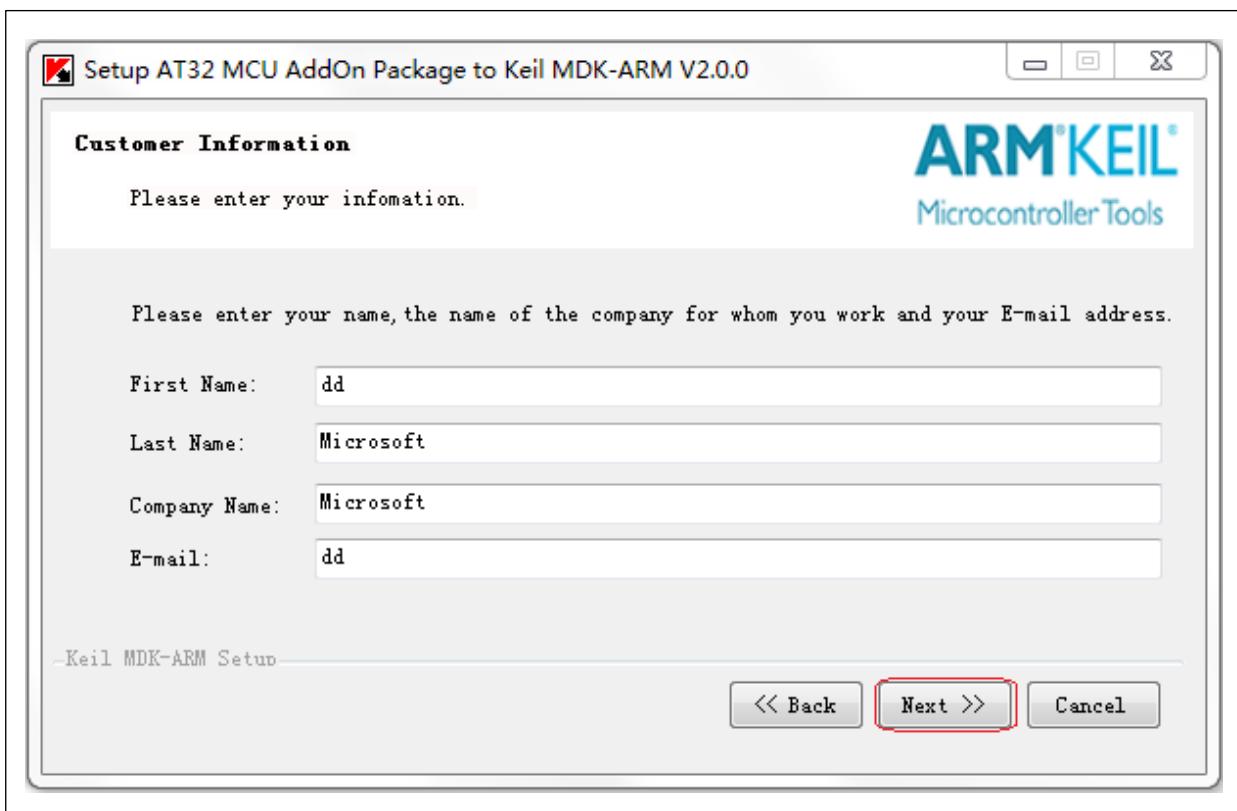
- ① Unzip *Keil4\_AT32MCU\_AddOn.zip*
- ② Click on *Keil4\_AT32MCU\_AddOn.exe*, and a dialog box pops up below (the specific version information is subject to the actual conditions).

Figure 6. Keil\_v4 Pack installation status



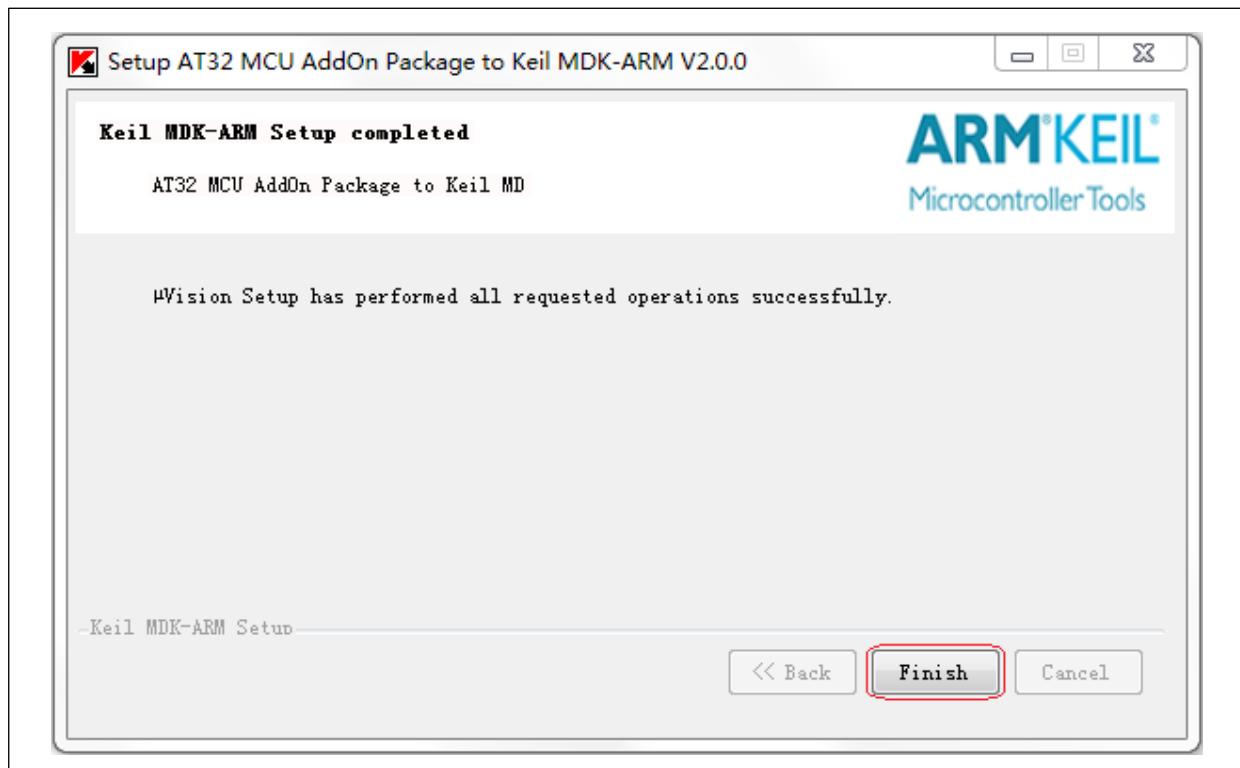
- ③ If the installation path of Keil\_v4 does not match the “Destination Folder”, click on “Browse” to select the actual correct path, then click on “Next”, as shown below.

Figure 7. Keil\_v4 Pack installation process



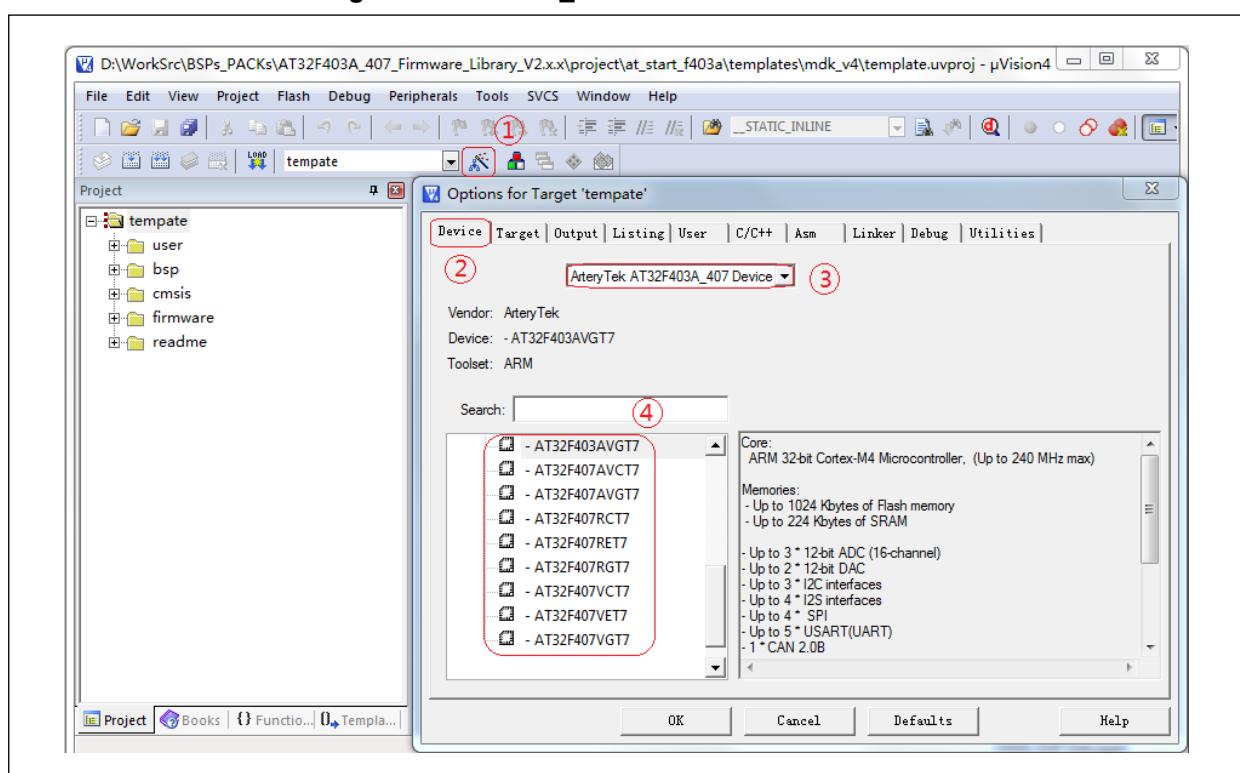
- ④ In the above “Customer Information” window, you can make some changes, but usually it is unnecessary. Then click on “Next” to start installation. The installation result is as follows.

Figure 8. Keil\_v4 Pack installation complete



- ⑤ Click on “Finish”. To check whether Keil\_v4 Pack is installed successfully or not, follow the below steps:
  - Click on “wand”
  - Select “Device”
  - Select the desired pack file
  - View ArteryTek-related information.

Figure 9. View Keil\_v4 Pack installation status

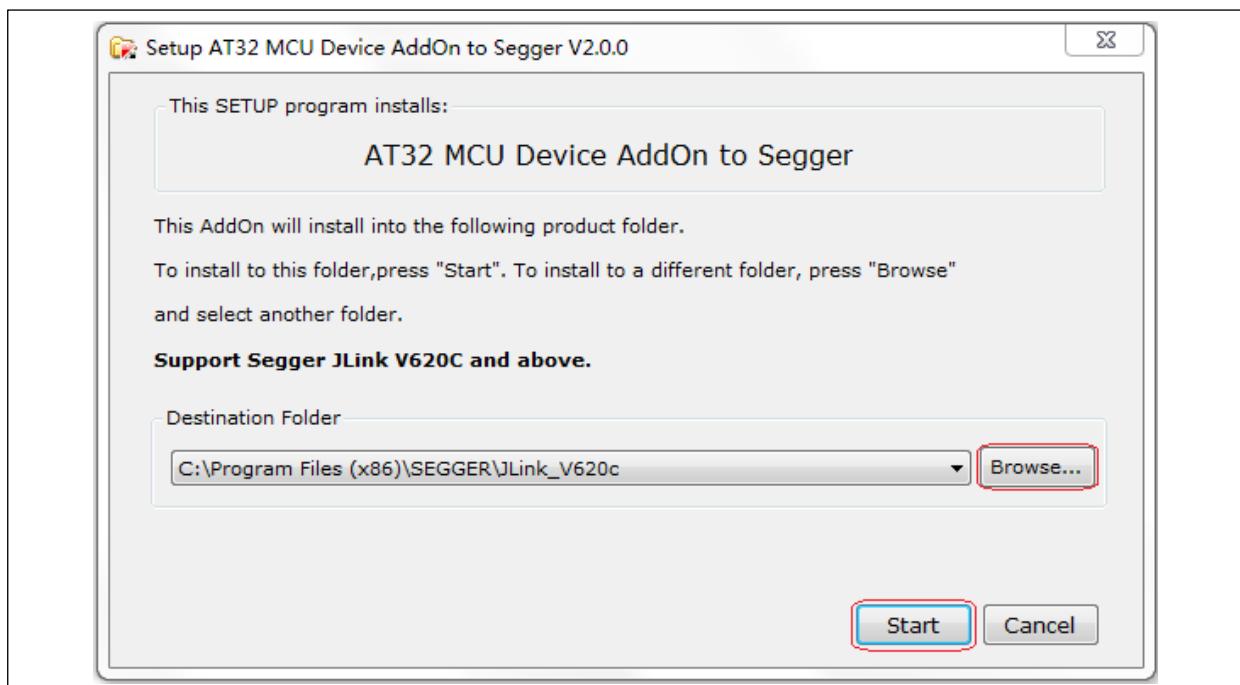


## 2.4 Segger Pack installation

**Segger\_AT32MCU\_AddOn.zip:** This is used to download J-Flash, follow the steps below to install:

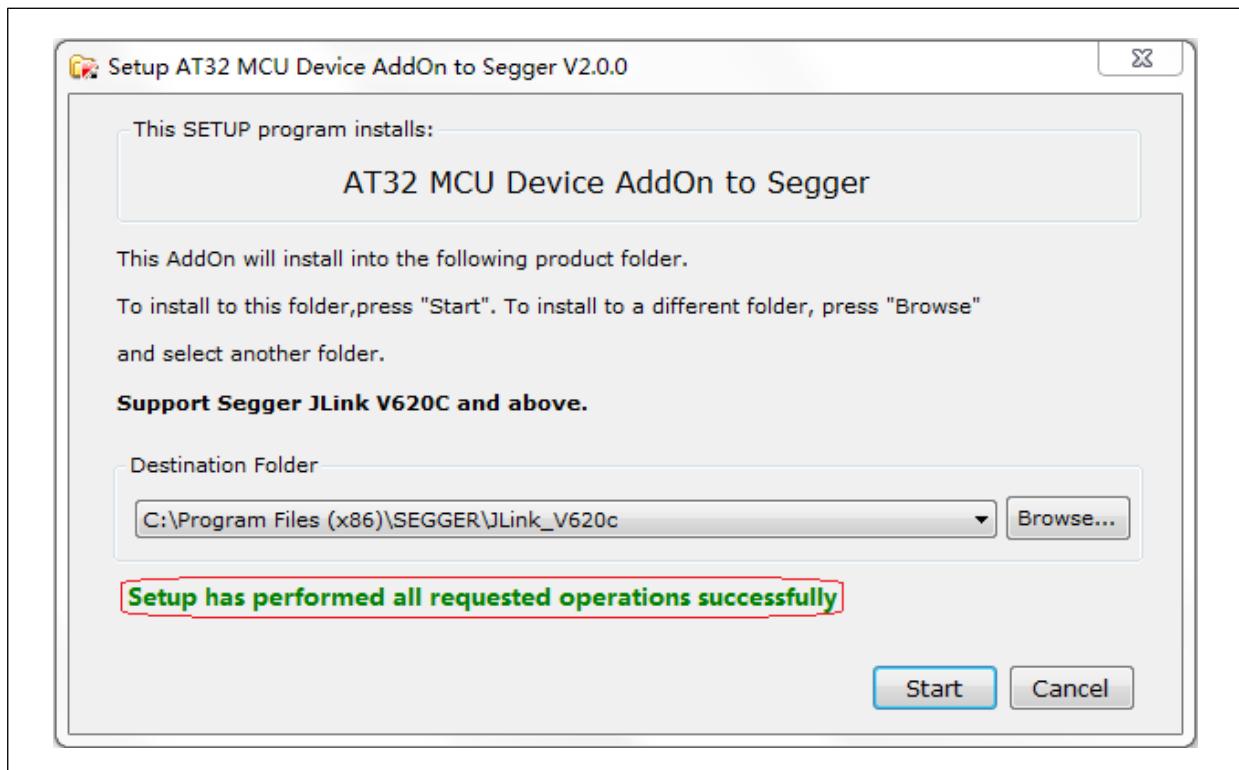
- ① Unzip Segger\_AT32MCU\_AddOn.zip
- ② Click on Segger\_AT32MCU\_AddOn.exe, and a dialog box pops up below (the specific version information is subject to the actual conditions)

Figure 10. Segger pack installation window



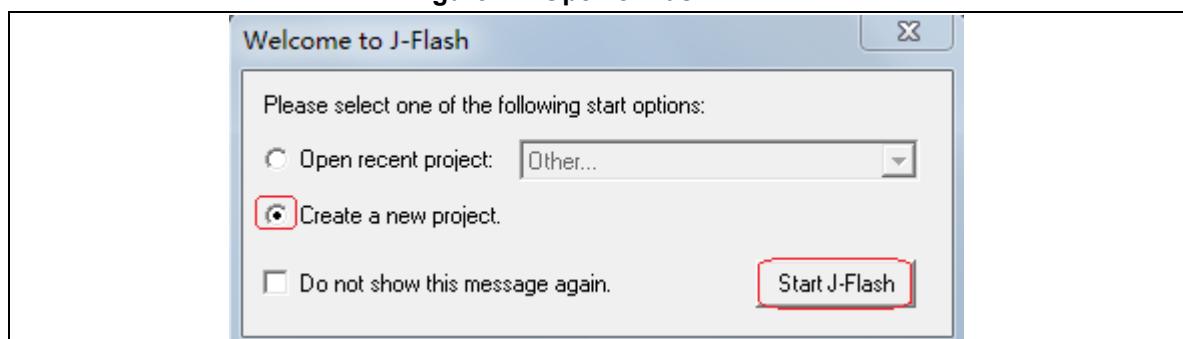
**Note:** If the installation path of Segger does not match the “Destination Folder”, click on “Browse” to select a correct path, then click on “Start”, as shown below.

Figure 11. Segger pack installation process



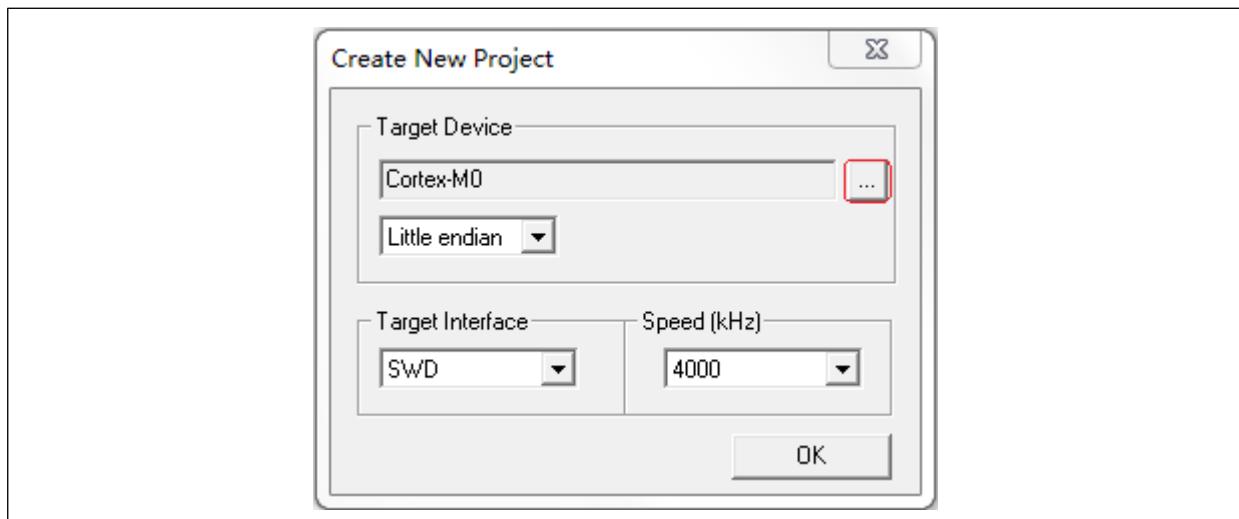
- ③ If the “Setup has performed all requested operations successfully” appears, it indicates successful installation. To check whether the installation is successful or not, follow the steps below:
- Open J-Flash.exe, a dialog box appears, tick “Create a new project” and click on “Start J-Flash”:

Figure 12. Open J-Flash



- After “Start J-Flash”, click on the check box under “Target Device”:

Figure 13. Create a new project using J-Flash



- Drag the scroll bar up and down in the check box. If the ArteryTek-related information and algorithm documents can be found, the installation is successful, as shown below:

Figure 14. View Device information

Manufacturer	Device	Core	Flash size	RAM size
ArteryTek	AT32F403_EXT_TYPE1_1MB	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE1_2MB	Cortex-M4	2048 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE1_4MB	Cortex-M4	4096 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE1_8MB	Cortex-M4	8192 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE2_16MB	Cortex-M4	16 MB	224 KB
ArteryTek	AT32F403_EXT_TYPE2_1MB	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE2_2MB	Cortex-M4	2048 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE2_4MB	Cortex-M4	4096 KB	224 KB
ArteryTek	AT32F403_EXT_TYPE2_8MB	Cortex-M4	8192 KB	224 KB
ArteryTek	AT32F403_UNIVERSAL_TYPE1_1...	Cortex-M4	128 KB + 16 MB	224 KB
ArteryTek	AT32F403_UNIVERSAL_TYPE1_2...	Cortex-M4	1024 KB + 16 MB	224 KB
ArteryTek	AT32F403_UNIVERSAL_TYPE2_1...	Cortex-M4	128 KB + 16 MB	224 KB
ArteryTek	AT32F403_UNIVERSAL_TYPE2_2...	Cortex-M4	1024 KB + 16 MB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	16 MB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	2048 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	4096 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	8192 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	16 MB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	2048 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	4096 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE1_REAMP...	Cortex-M4	8192 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE2_REAMP...	Cortex-M4	16 MB	224 KB
ArteryTek	AT32F403A_EXT_TYPE2_REAMP...	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE2_REAMP...	Cortex-M4	2048 KB	224 KB
ArteryTek	AT32F403A_EXT_TYPE2_RFAMP...	Cortex-M4	1024 KB	224 KB
ArteryTek	AT32F403A_FXT_TYPE2_RFAMP	Cortex-M4	2048 KB	224 KB

OK Cancel

### 3 Flash algorithm file

Flash algorithm files are included in the Pack for online download through IDE tools such as KEIL/IAR. This section describes how to use Flash algorithm files.

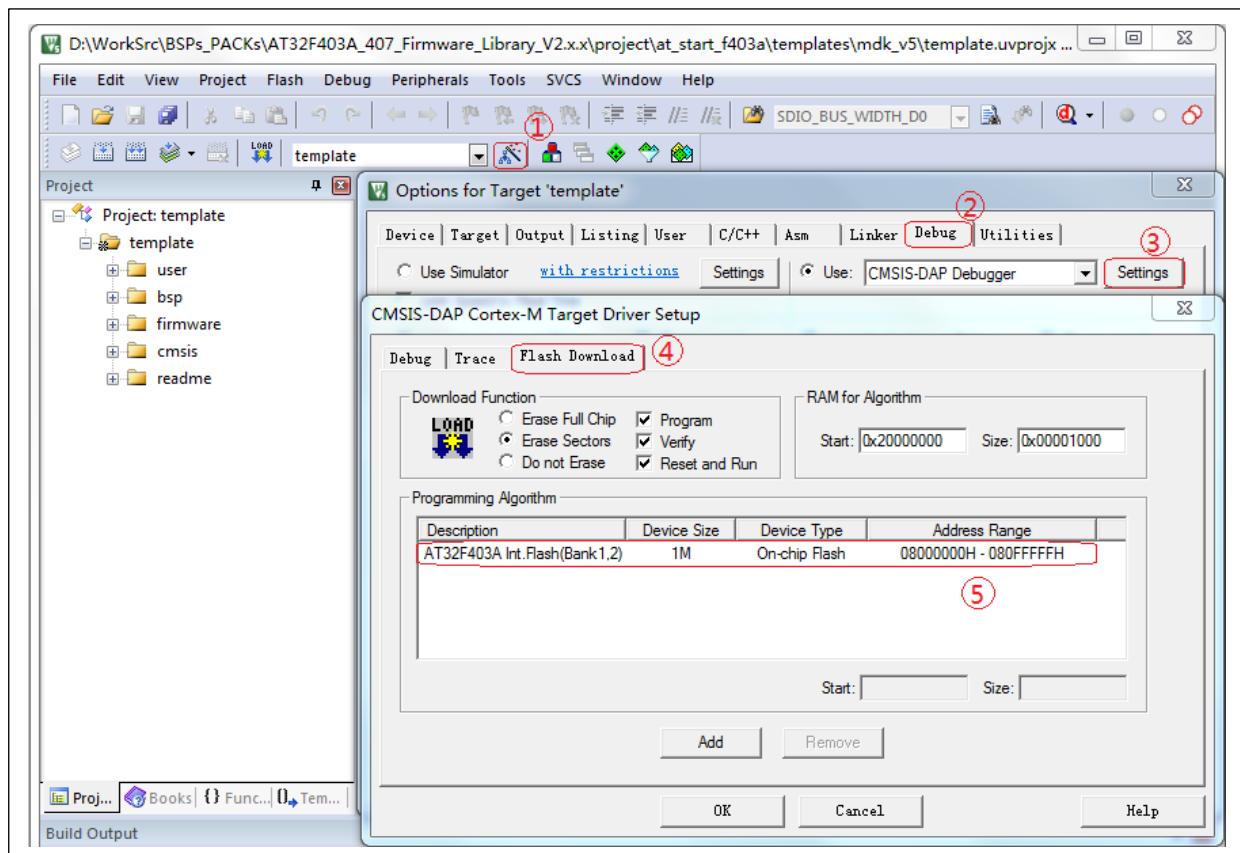
Note: AT32 MCUs have similar Flash algorithms, and this section uses AT32F403A as an example.

#### 3.1 How to use Keil algorithm file

Common IDE tools such as Keil\_v4 and Keil\_v5 adopt a similar method to select and use the algorithm files. Here we take Keil\_v5 as an example.

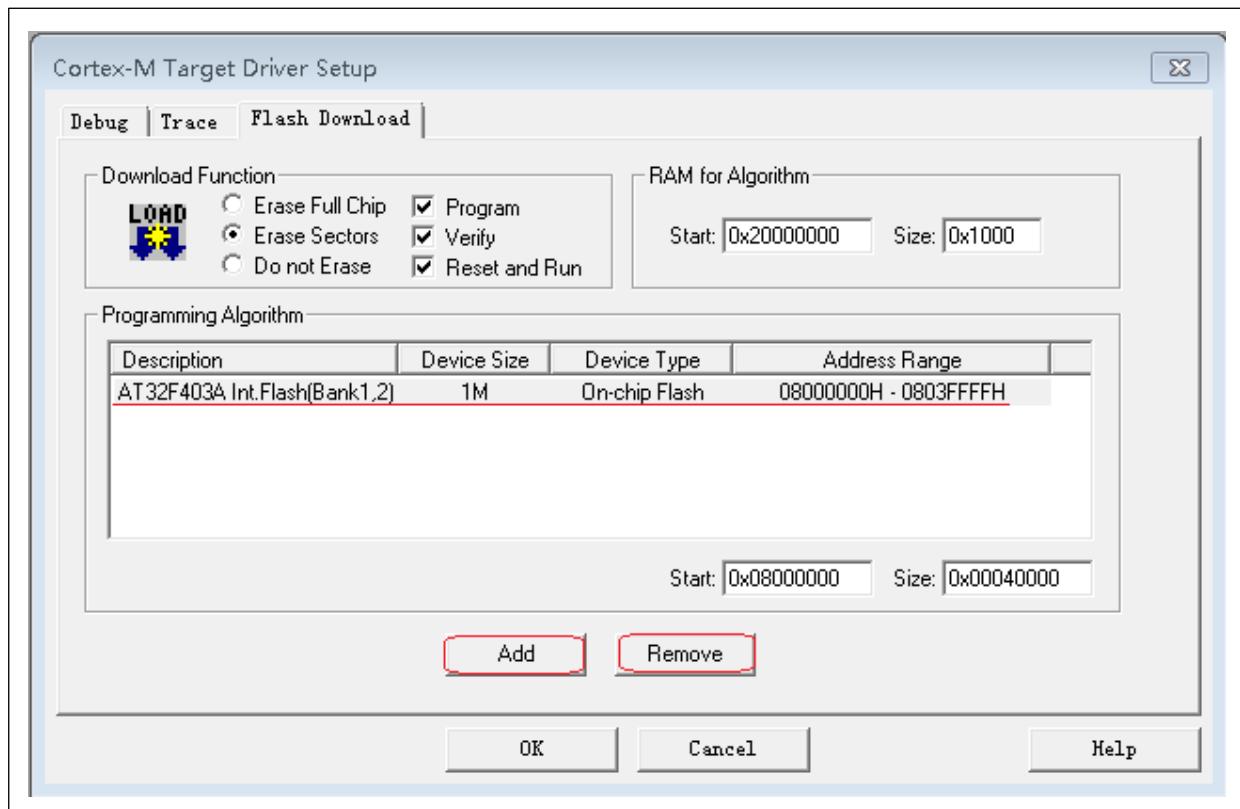
After creating a Keil IDE development tool project, the user can start Debug configuration and select the Flash algorithms. Go to *wand* → *Debug* → *Settings* → *Flash Download*, as shown below:

Figure 15. Keil algorithm file settings



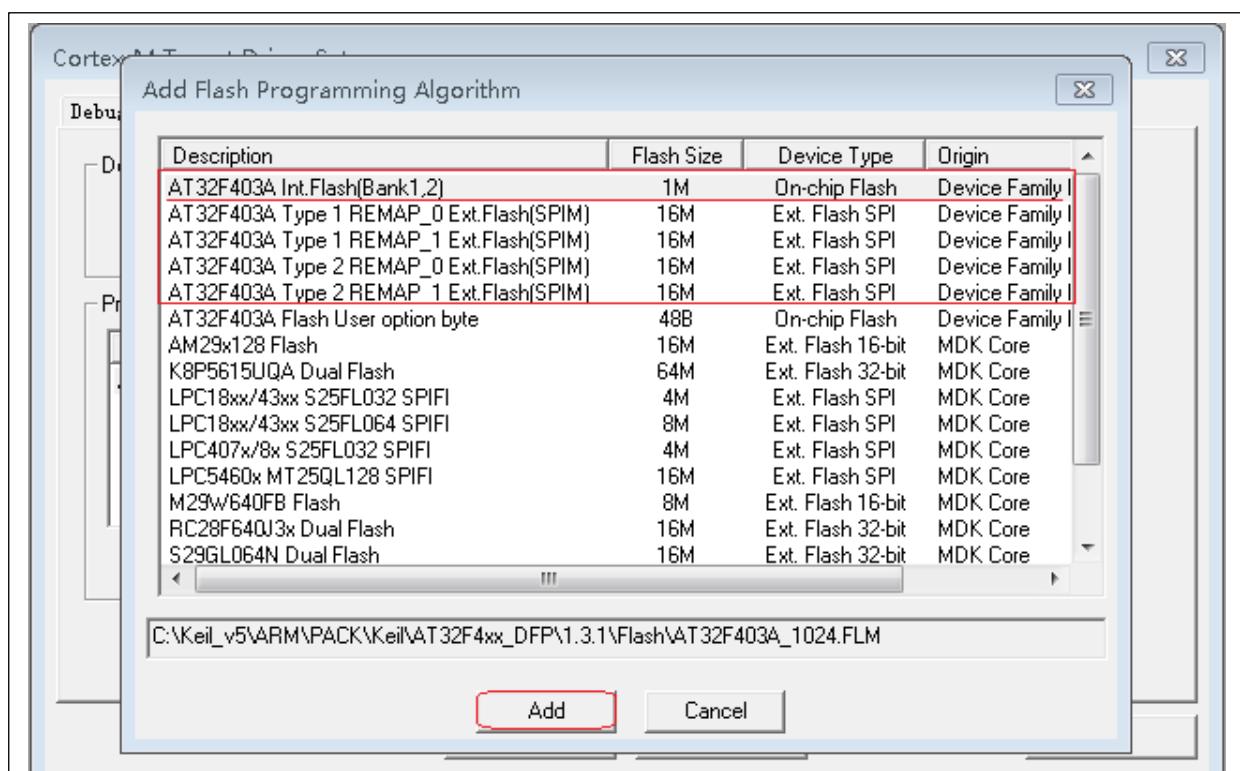
In this example, the selected Flash algorithm file is the default one. To change or remove it, click on this algorithm file, then click on *Add* or *Remove*. If the selected algorithm does not match the MCU, please follow the method below to modify.

Figure 16. Keil algorithm file configuration



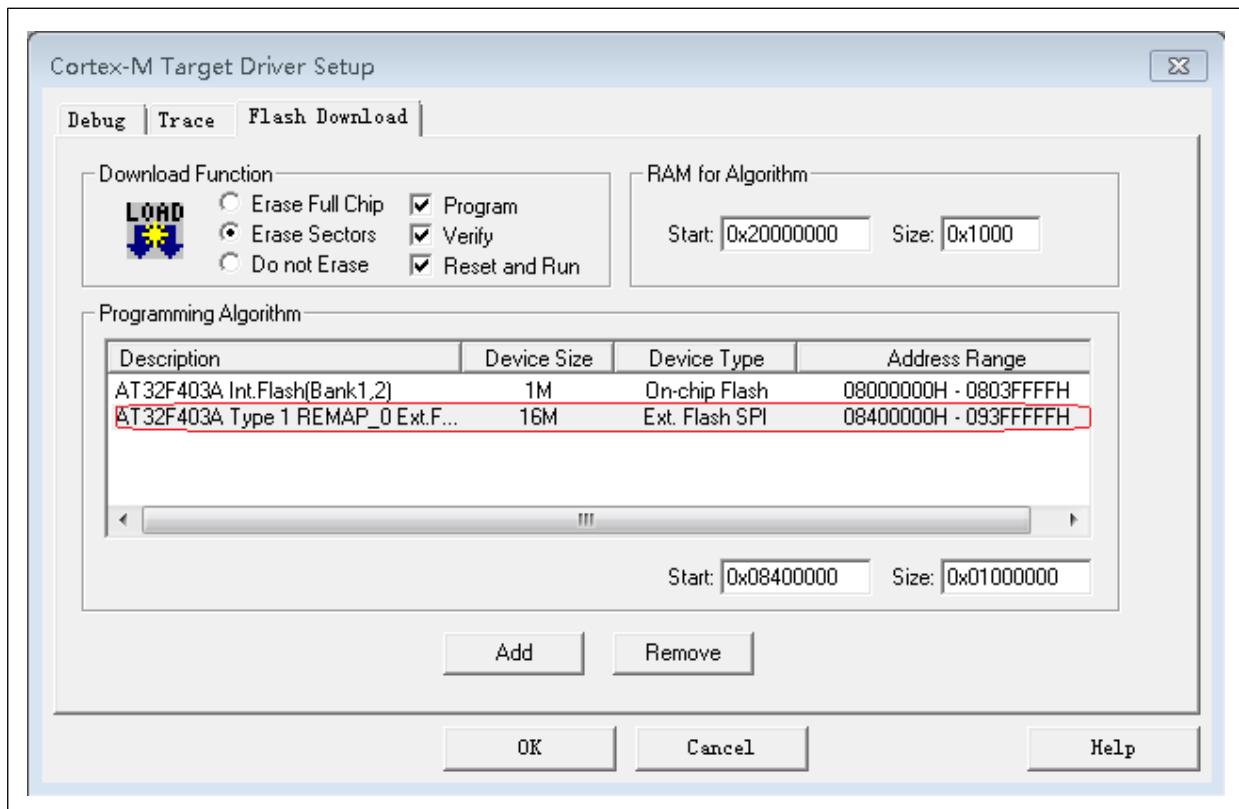
Click on *Remove* to remove the existing algorithm from the configuration, then click on *Add* to view the algorithm files associated with a MCU model and select them, as shown below:

Figure 17. Select algorithm files using Keil



After selection, click on *Add* to add the selected algorithm files into the current configuration. For example, a new SPIFI algorithm is added into the project.

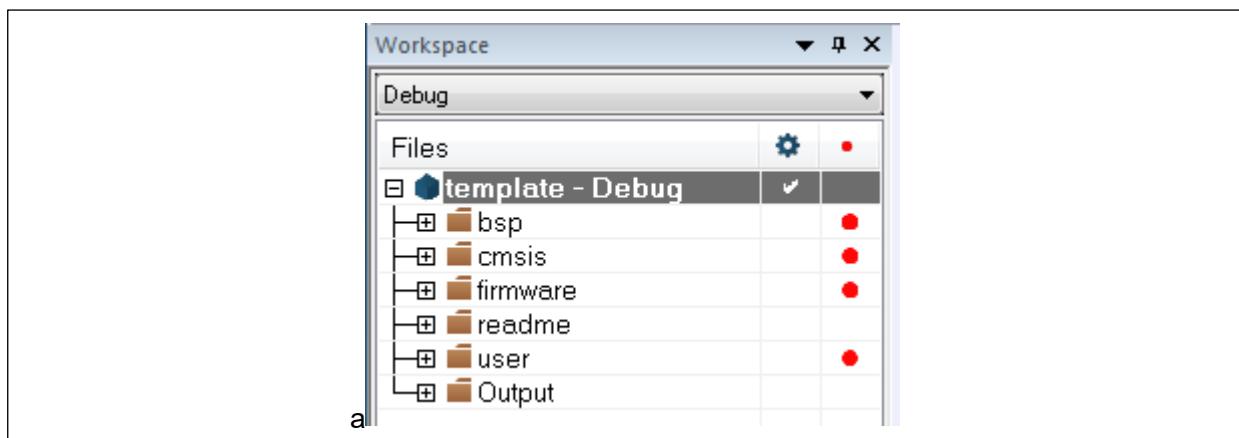
Figure 18. Add algorithm files using Keil



## 3.2 How to use IAR algorithm files

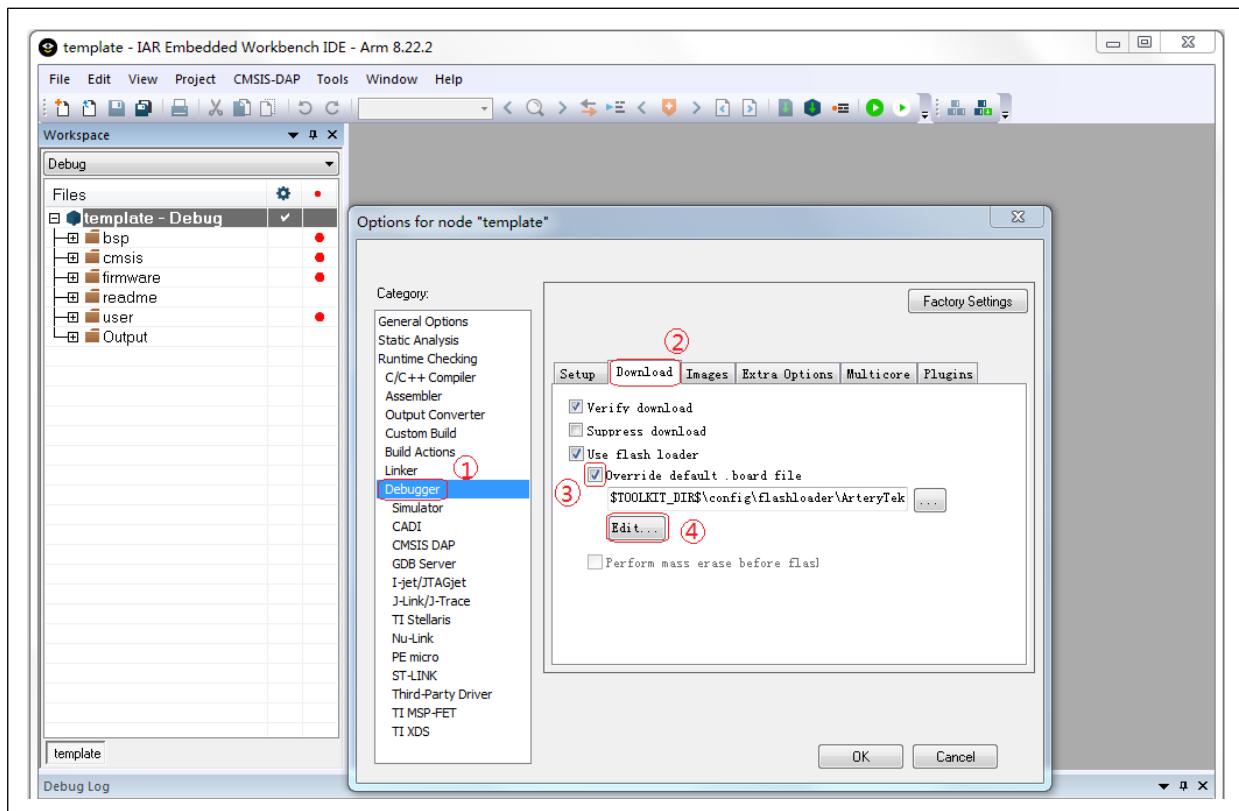
In IAR environment, the Flash algorithm files are automatically selected according to the selected MCU model during a new project configuration. To configure/modify an algorithm file manually, right-click on the file name (after an IAR project is created) in the following gray box:

Figure 19. IAR project name



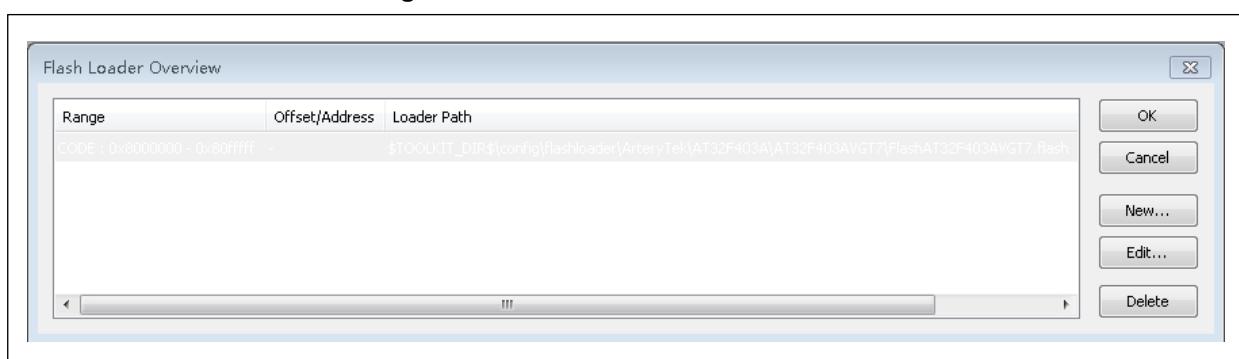
Go to *Options*—>*Debugger*—>*Download*—> Tick *Override default .board file*—> Click on *Edit*, as shown below:

Figure 20. IAR algorithm file configuration



Then the following window will be displayed.

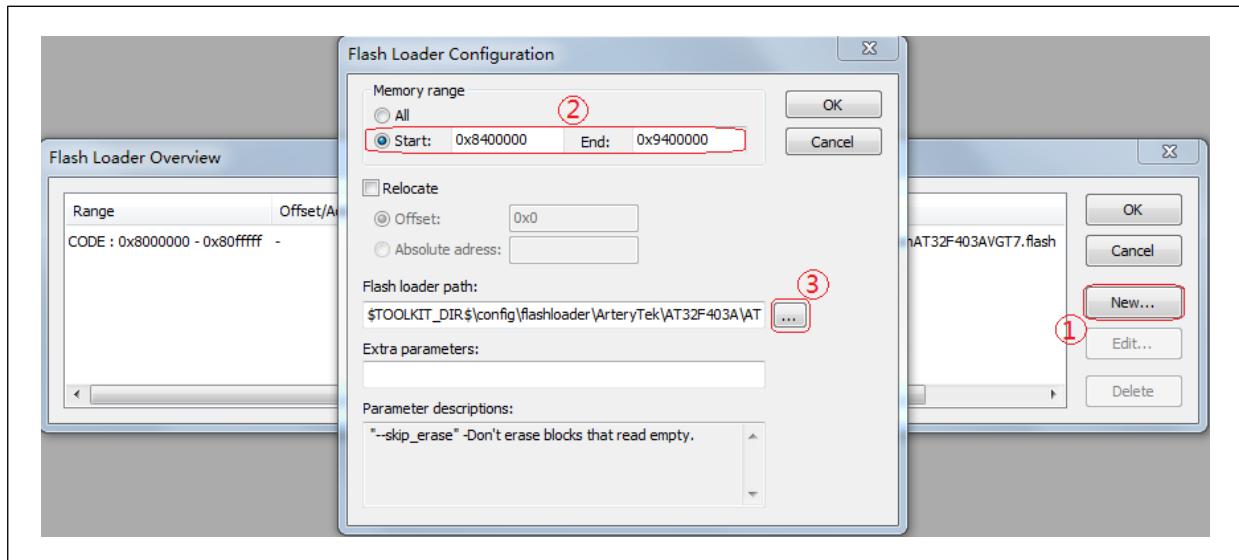
Figure 21. IAR Flash Loader Overview



Flash algorithm configuration is designated by default after selecting a MCU part number. To modify it, click on *New/Edit/Delete*.

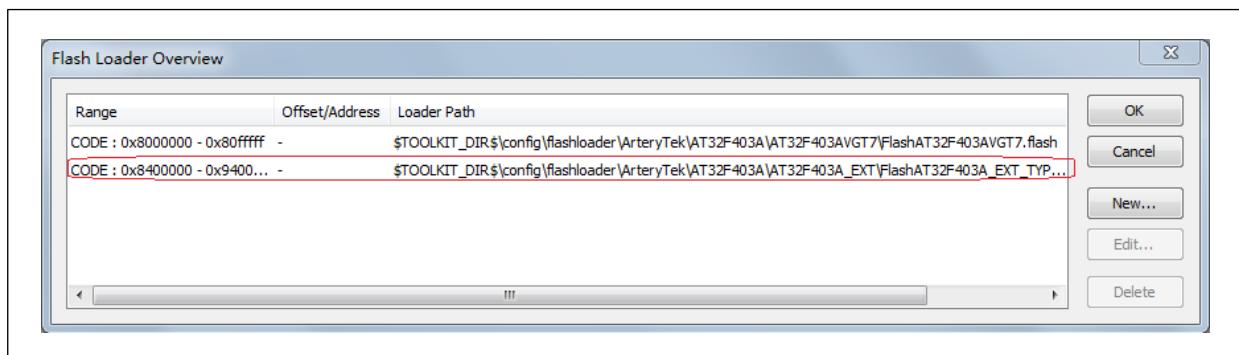
For example, click on *New* —> *Memory range* —> Select a Flash algorithm file, as shown below:

Figure 22. IAR Flash Loader configuration



This example shows how to add a SPIM Flash algorithm file. The user needs to select the corresponding MCU part number and a correct Flash algorithm file. The selected Flash algorithm configuration file is installed into IAR development environment using IAR\_AT32F4xx\_AddOn tool. After a successful configuration, a new SPIM Flash algorithm is shown below:

Figure 23. IAR Flash Loader configuration success



### 1. Description of SPIM algorithms

Some Artery MCUs support Bank3 (refer to the Reference Manual or Datasheet on Artery official website for details), which can be used as an expansion of Flash memory in case of insufficient internal Flash or special application requirements. When the compiling addresses of some code or data are stored in the SPIM, these algorithm files are used for external Flash programming during online IDE tool download.

Naming rules of Artery SPIM algorithm file: AT32F4xxTypeNREMAP\_P Ext.Flash

N=1,2

P=0,1

**TYPEN:** External SPI Flash, select it according to the external Flash type and part number. Refer to the FLASH\_SELECT register section of the corresponding MCU Reference Manual.

**REMAP\_P:** Select multiplex-function MCU SPIM PIN, select it according to the connection method of pins connected to external Flash. Refer to the external SPIF remapping section in the corresponding MCU reference manual.

REMAP0:EXT\_SPIF\_GRMP=000

REMAP1:EXT\_SPIF\_GRMP=001

## 4 BSP introduction

### 4.1 Quick start

#### 4.1.1 Template project

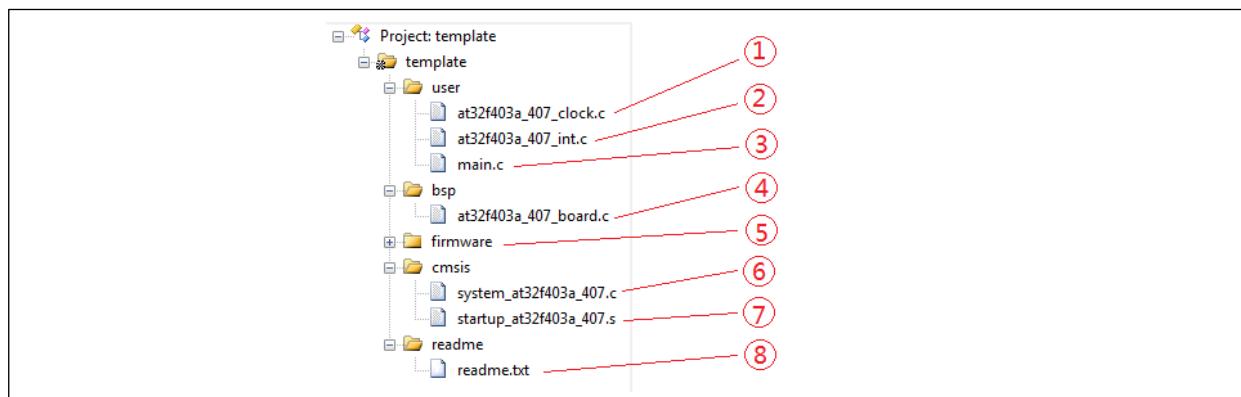
Artery firmware library BSP comes with a series of template projects built around Keil and IAR. For example, the template project of AT32F403A/407 is located in `AT32F403A_407_Firmware_Library_V2.x.x/project/at_start_xxx/templates`.

**Figure 24. Template content**

iar_v6.10	21/05/24 16:03	文件夹
iar_v7.4	21/05/24 16:03	文件夹
iar_v8.2	21/05/24 16:03	文件夹
inc	21/05/24 16:03	文件夹
mdk_v4	21/05/24 16:03	文件夹
mdk_v5	21/05/24 16:03	文件夹
src	21/05/24 16:03	文件夹
readme.txt	21/05/21 11:15	TXT 文件

The above template project includes various versions such as Keil\_v5, Keil\_v4, IAR\_6.10, IAR\_7.4 and IAR\_8.2. Of those, “inc” and “src” folders contain header files and source code files. Open a corresponding folder and click on the corresponding file to open an IDE project. Figure 25 presents an example of Keil\_v5 template project (its details and version are subject to the actual firmware library).

**Figure 25. Keil\_v5 template project example**



The contents in a project include: (using AT32F403A/407 as an example, other products are similar)

- ① at32f403a\_407\_clock.c (clock configuration file) defines the default clock frequency and clock paths
- ② at32f403a\_407\_int.c (interrupt file) contains some interrupt handling codes
- ③ main.c contains the main code files.
- ④ at32f403a\_407\_board.c (board configuration file) contains common hardware configurations such as buttons and LEDs on the AT-START-Evaluation Board.
- ⑤ at32f403a\_407\_xx.c under firmware folder contains driver files of on-chip peripherals
- ⑥ system\_at32f403a\_407.c is the system initialization file
- ⑦ startup\_at32f403a\_407.s is a startup file
- ⑧ readme.txt is a readme file, containing functional description and configuration information

*Note:* AT32 MUCs share similar BSP usage method, and this section uses AT32F403A as an example.

## 4.1.2 BSP macro definitions

- ① To create a project, it is necessary to enable a startup code (startup\_at32f403a\_407.s) and open the appropriate macro definitions according to MCU part number before compiling code. Table 1 presents the correspondence between the MCU and their macro definitions.

**Table 1. Summary of macro definitions**

MCU part numbers	Macro definitions	PINs	Flash size (KB)
AT32F403ACCT7	AT32F403ACCT7	48	256
AT32F403ACET7	AT32F403ACET7	48	512
AT32F403ACGT7	AT32F403ACGT7	48	1024
AT32F403ACCU7	AT32F403ACCU7	48	256
AT32F403ACEU7	AT32F403ACEU7	48	512
AT32F403ACGU7	AT32F403ACGU7	48	1024
AT32F403ARCT7	AT32F403ARCT7	64	256
AT32F403ARET7	AT32F403ARET7	64	512
AT32F403ARGT7	AT32F403ARGT7	64	1024
AT32F403AVCT7	AT32F403AVCT7	100	256
AT32F403AVET7	AT32F403AVET7	100	512
AT32F403AVGT7	AT32F403AVGT7	100	1024
AT32F407RCT7	AT32F407RCT7	64	256
AT32F407RET7	AT32F407RET7	64	512
AT32F407RGT7	AT32F407RGT7	64	1024
AT32F407VCT7	AT32F407VCT7	100	256
AT32F407VET7	AT32F407VET7	100	512
AT32F407VGT7	AT32F407VGT7	100	1024
AT32F407AVCT7	AT32F407AVCT7	100	256
AT32F407AVGT7	AT32F407AVGT7	100	1024

- ② In the header file (at32f403a\_407.h), USE\_STDPERIPH\_DRIVER (macro definition) is used to determine whether the Keil RTE feature is used or not. Enabling this definition while Keil RTE is unused can prevent some versions of Keil-MDK from opening \_RTE\_ accidentally.
- ③ The configuration header file (at32f403a\_407\_conf.h) defines macro definitions that enable peripherals. The file can be used to control the use of peripherals. The peripherals can be disabled simply by masking \_MODULE\_ENABLED pertaining to peripherals, as shown below:

**Figure 26. Peripheral enable macro definitions**

```
#define CRM_MODULE_ENABLED
#define TMR_MODULE_ENABLED
#define RTC_MODULE_ENABLED
#define BPR_MODULE_ENABLED
#define GPIO_MODULE_ENABLED
#define I2C_MODULE_ENABLED
#define USART_MODULE_ENABLED
#define FWC_MODULE_ENABLED
#define CAN_MODULE_ENABLED
#define ADC_MODULE_ENABLED
#define DAC_MODULE_ENABLED
#define SPI_MODULE_ENABLED
#define DMA_MODULE_ENABLED
#define DEBUG_MODULE_ENABLED
#define FLASH_MODULE_ENABLED
#define CRC_MODULE_ENABLED
#define WWDI_MODULE_ENABLED
#define NWDI_MODULE_ENABLED
#define EXINT_MODULE_ENABLED
#define SDIO_MODULE_ENABLED
#define XMC_MODULE_ENABLED
#define USB_MODULE_ENABLED
#define ACC_MODULE_ENABLED
#define MISC_MODULE_ENABLED
#define ENMAC_MODULE_ENABLED
```

at32f403a\_407\_conf.h also defines the HEXT\_VALUE (high-speed external clock value), which should be modified accordingly when changing an external high-speed crystal oscillator.

- ④ The system clock configuration file (at32f403a\_407\_clock.c/h) defines the default system clock frequency and clock paths. The user, if needed, can customize the frequency multiplication process and factors, or generate corresponding clock configuration files using the clock configuration host of ArteryTek.

## 4.2 BSP specifications

The subsequent sections give a description of BSP specifications.

### 4.2.1 List of abbreviations for peripherals

Table 2. List of abbreviations for peripherals

Abbreviations	Description
ADC	Analog-to-digital converter
BPR	Battery powered register
CAN	Controller area network
CRC	CRC calculation unit
CRM	Clock and reset manage
DAC	Digital-to analog converter
DMA	Direct memory access
DEBUG	Debug
EXINT	External interrupt/event controller
GPIO	General-purpose I/Os
IOMUX	Multiplexed I/Os
I2C	Inter-integrated circuit interface
NVIC	Nested vectored interrupt controller
PWC	Power controller
RTC	Real-time clock
SPI	Serial peripheral interface
I2S	Inter-IC Sound
SysTick	System tick timer
TMR	Timer
USART	Universal synchronous asynchronous receiver transmitter
WDT	Watchdog timer
WWDT	Window watchdog timer
XMC	External memory controller

## 4.2.2 Naming rules

The naming rules for BSP are described as follows:

“ip” indicates an abbreviation of a peripheral, for example, ADC, TMR, GPIO, etc, regardless of upper and lower case letters, such as, adc,tmr,gpio...

- **Source code file**

The file name starts with “at32fxxx\_ip.c”, for example, at32f403a\_407\_adc.c

- **Header file**

The file name starts with “at32fxxx\_ip.h”, such as, at32f403a\_407\_adc.h

- **Constant**

If it is used in a single one file, the constant is then defined in this file; if it is used in multiple files, the constant is defined in corresponding header file.

All constants are in written in English capital letters.

- **Variable**

If it is used in a single one file, the variable is then defined in this file; if it is used in multiple files, the variable is declared with extern in the corresponding header file.

- **Naming rules for functions**

The peripheral functions are named based on the rule of “peripheral abbreviation\_attribute\_action” or “peripheral abbreviation\_action”.

The commonly used functions are as follows:

Function type	Naming rule	Example
Peripheral reset	ip_reset	such as, adc_reset
Peripheral enable	ip_enable	adc_enable
Peripheral structure parameter initialize	ip_default_para_init	spi_default_para_init
Peripheral initialize	ip_init	spi_init
Peripheral interrupt enable	ip_interrupt_enable	adc_interrupt_enable
Peripheral flag get	ip_flag_get	adc_flag_get
Peripheral flag clear	ip_flag_clear	adc_flag_clear

## 4.2.3 Encoding rules

This section describes the encoding rules related to firmware function library.

Type of variables:

```
typedef int32_t INT32;
typedef int16_t INT16;
typedef int8_t INT8;
typedef uint32_t UINT32;
typedef uint16_t UINT16;
typedef uint8_t UINT8;

typedef int32_t s32;
typedef int16_t s16;
typedef int8_t s8;

typedef const int32_t sc32; /*!< read only */
typedef const int16_t sc16; /*!< read only */
typedef const int8_t sc8; /*!< read only */

typedef __IO int32_t vs32;
typedef __IO int16_t vs16;
typedef __IO int8_t vs8;

typedef __I int32_t vsc32; /*!< read only */
typedef __I int16_t vsc16; /*!< read only */
typedef __I int8_t vsc8; /*!< read only */

typedef uint32_t u32;
typedef uint16_t u16;
typedef uint8_t u8;

typedef const uint32_t uc32; /*!< read only */
typedef const uint16_t uc16; /*!< read only */
typedef const uint8_t uc8; /*!< read only */

typedef __IO uint32_t vu32;
typedef __IO uint16_t vu16;
typedef __IO uint8_t vu8;

typedef __I uint32_t vuc32; /*!< read only */
typedef __I uint16_t vuc16; /*!< read only */
typedef __I uint8_t vuc8; /*!< read only */
```

#### 4.2.3.1 Flag type

```
typedef enum {RESET = 0, SET = !RESET} flag_status;
```

#### 4.2.3.2 Function status type

```
typedef enum {FALSE = 0, TRUE = !FALSE} confirm_state;
```

#### 4.2.3.3 Error status type

```
typedef enum {ERROR = 0, SUCCESS = !ERROR} error_status;
```

#### 4.2.3.4 Peripheral type

##### ① Peripherals

Define the base address of peripheral in the at32fxxx\_ip.h, for example, in the at32f403a\_407.h:

```
#define ADC1_BASE (APB2PERIPH_BASE + 0x2400)  
#define ADC2_BASE (APB2PERIPH_BASE + 0x2800)
```

Define the type of a peripheral in the at32fxxx\_ip.h, for example, in the at32f403a\_407\_adc.h:

```
#define ADC1 ((adc_type *)ADC1_BASE)  
#define ADC2 ((adc_type *)ADC2_BASE)
```

##### ② Peripheral registers and bits

Define the type of a peripheral in the at32fxxx\_ip.h, for example, in the at32f403a\_407\_adc.h

```
/*  
 * @brief type define adc register all  
 */  
typedef struct  
{  
  
    /**  
     * @brief adc sts register, offset:0x00  
     */  
    union  
    {  
        __IO uint32_t sts;  
        struct  
        {  
            __IO uint32_t vmor :1; /* [0] */  
            __IO uint32_t cce :1; /* [1] */  
            __IO uint32_t pcce :1; /* [2] */  
            __IO uint32_t pccs :1; /* [3] */  
            __IO uint32_t occs :1; /* [4] */  
            __IO uint32_t reserved1 :27; /* [31:5] */  
        } sts_bit;  
    };  
};  
...
```

```
...
...
/**  
 * @brief adc odt register, offset:0x4C  
 */  
union  
{  
    __IO uint32_t odt;  
    struct  
    {  
        __IO uint32_t odt          :16; /* [15:0] */  
        __IO uint32_t adc2odt     :16; /* [31:16] */  
    } odt_bit;  
};  
  
} adc_type;
```

③ Examples of peripheral register access

Read peripheral	i = ADC1-> ctrl1;
Write peripheral	ADC1-> ctrl1 = i;
Read bit 5 in bit-field mode	i = ADC1-> ctrl1. cceien;
Write 1 to bit 5 in bit-field mode	ADC1-> ctrl1. cceien= TRUE;
Write 1 to bit 5	ADC1-> ctrl1  = 1<<5;
Write 0 to bit 5	ADC1-> ctrl1&= ~(1<<5) ;

## 4.3 BSP structure

### 4.3.1 BSP folder structure

BSP (Board Support Package) structure is shown in Figure 27.

Figure 27. BSP folder structure

document	21/05/18 10:32	文件夹
libraries	21/05/18 10:32	文件夹
middlewares	21/05/18 10:32	文件夹
project	21/05/18 10:32	文件夹
utilities	21/05/14 11:35	文件夹

#### Document:

- AT32Fxxx firmware library BSP&Pack user guide.pdf: refer to BSP/Pack user manual
- ReleaseNotes\_AT32F403A\_407\_Firmware\_Library.pdf: document revision history

#### Libraries:

- **Drivers:** driver library for peripherals  
Src folder: low-level driver source file for peripherals, such as, at32fxxx\_ip.c  
Inc folder: low-level driver head file for peripherals, such as, at32fxxx\_ip.h
- **Cmsis:** Core-related files  
cm4 folder: core-related files, including cortex-m4 library, system initialization file, startup file, etc.  
dsp folder: dsp-related files

#### Middlewares:

Third-party software or public protocols, including USB protocol layer driver, network protocol driver, operating system source code, etc.

#### Project:

Examples: demo

Templates: template project, including Keil4, keil5, IAR6, IAR7, IAR8 and eclipse\_gcc

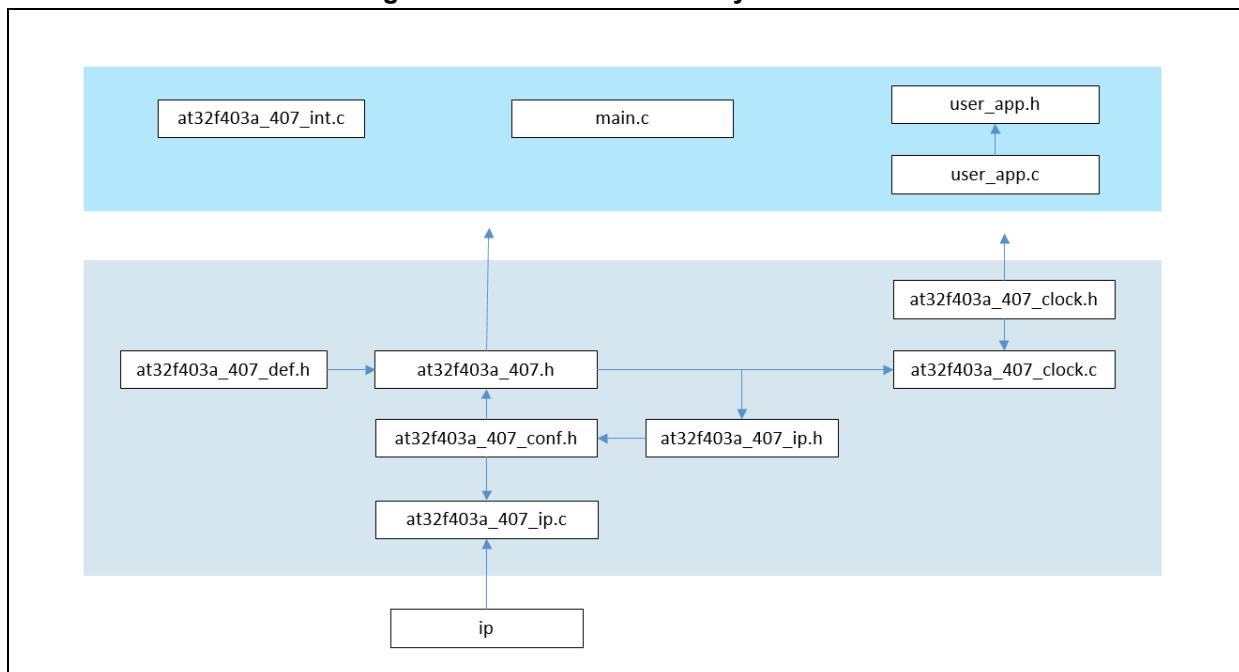
#### Utilities:

Store application cases

### 4.3.2 BSP function library structure

Figure 28 shows the architecture of BSP function library.

**Figure 28. BSP function library structure**



BSP function library files are described in Table 3.

**Table 3. Summary of BSP function library files**

File name	Description
at32f403a_407_conf.h	Macro definition for peripheral enable, and external high-speed clock HEXT_VALUE
main.c	Main function
at32f403a_407_ip.c	Driver source file for a peripheral, for example, at32f403a_407_adc.c
at32f403a_407_ip.h	Driver header file for a peripheral, for example, at32f403a_407_adc.h
at32f403a_407.h	In the header file (at32f403a_407.h), the definition USE_STDPERIPH_DRIVER is used to determine whether the Keil RTE is used or not. Enabling the definition while Keil RTE is unused can prevent Keil-MDK from enabling _RTE_ accidentally.
at32f403a_407_clock.c	This is a clock configuration file used to configure default clock frequency and clock path
at32f403a_407_clock.h	This is a clock config header file.
at32f403a_407_int.c	This is a source file for interrupt functions that programs interrupt handling code.
at32f403a_407_int.h	This is a header file for interrupt functions
at32f403a_407_misc.c	This is a source file for other configurations, such as, nvic configuration function, systick clock source selection.
at32f403a_407_misc.h	This is header file for other configurations
startup_at32f403a_407.s	This is a startup file.

### 4.3.3 Initialization and configuration for peripherals

This section describes how to initialize and configure peripherals using GPIO as an example.

#### GPIO initialization

- Step 1: Define the structure `gpio_init_type`, for example, `gpio_init_type gpio_init_struct;`
- Step 2: Enable GPIO clock using the function `crm_periph_clock_enable`
- Step 3: De-initialize the structure `gpio_init_struct` to allow the values of other members (mostly default values) to be correctly written, for example,  
`gpio_default_para_init(&gpio_init_struct);`
- Step 4: Configure member of the structure, and write structure parameters into GPIO registers through the `gpio_init`, for example,

```
gpio_init_struct.gpio_pins = GPIO_PINS_2 | GPIO_PINS_3;  
gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;  
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;  
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;  
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;  
gpio_init(GPIOA, &gpio_init_struct);
```

For more information on peripheral initialization procedure, refer to the section of peripherals of the reference manual, and the section of peripherals of the [AT32Fxxx\\_Firmware\\_Library\\_V2.x.x.zip\project\at\\_start\\_fxxx\examples](#).

### 4.3.4 Peripheral functions description format

Table 4. Function format description for peripherals

Name	Description
Function name	The name of a peripheral function
Function prototype	Prototype declaration
Function description	Brief description of how the function is executed
Input parameter (n)	Description of the input parameters
Output parameter (n)	Description of the output parameters
Return value	Value returned by the function
Required preconditions	Requirements before calling the function
Called functions	Other library functions called

## 5 AT32F403 peripheral library functions

### 5.1 Analog-to-digital converter (ADC)

ADC register structure adc\_type is defined in the “at32f403\_acc.h”:

```
/**  
 * @brief type define adc register all  
 */  
typedef struct  
{  
    .....  
} adc_type;
```

The table below gives a list of the ADC registers.

Table 5. Summary of ADC registers

Register	Description
sts	ADC status register
ctrl1	ADC control register 1
ctrl2	ADC control register 2
spt1	ADC sample time register 1
spt2	ADC sample time register 2
pcdto1	ADC preempted channel data offset register 1
pcdto2	ADC preempted channel data offset register 2
pcdto3	ADC preempted channel data offset register 3
pcdto4	ADC preempted channel data offset register 4
vmhb	ADC voltage monitor high boundary register
vmlb	ADC voltage monitor low boundary register
osq1	ADC ordinary sequence register 1
osq2	ADC ordinary sequence register 2
osq3	ADC ordinary sequence register 3
psq	ADC preempted sequence register
pdt1	ADC preempted data register 1
pdt2	ADC preempted data register 2
pdt3	ADC preempted data register 3
pdt4	ADC preempted data register 4
odt	ADC ordinary data register

The table below gives a list of ADC library functions.

**Table 6. Summary of ADC library functions**

Function name	Description
adc_reset	Reset all ADC registers to their reset values
adc_enable	Enable A/D converter
adc_combine_mode_select	Master/slave mode selection
adc_base_default_para_init	Define an initial value for adc_base_struct
adc_base_config	Configure ADC registers with the initialized parameters of the adc_base_struct
adc_dma_mode_enable	Enable DMA transfer for ordinary group
adc_interrupt_enable	Enable the selected ADC event interrupt
adc_calibration_init	Initialization calibration
adc_calibration_init_status_get	Get initialization calibration status
adc_calibration_start	Start calibration
adc_calibration_status_get	Get calibration status
adc_voltage_monitor_enable	Enable voltage monitoring for ordinary/preempted channels and a single channel
adc_voltage_monitor_threshold_value_set	Set the threshold of voltage monitoring
adc_voltage_monitor_single_channel_select	Select a single channel for voltage monitoring
adc_ordinary_channel_set	Configure ordinary channels, including channel selection, conversion sequence number and sampling time
adc_preempt_channel_length_set	Configure the length of preempted group conversion sequence
adc_preempt_channel_set	Configure preempted channels, including channel selection, conversion sequence number and sampling time
adc_ordinary_conversion_trigger_set	Enable trigger mode and trigger event selection for ordinary conversion
adc_preempt_conversion_trigger_set	Enable trigger mode and trigger event selection for preempted conversion
adc_preempt_offset_value_set	Set data offset for preempted conversion
adc_ordinary_part_count_set	Set the number of ordinary channels for each triggered conversion in partition mode
adc_ordinary_part_mode_enable	Enable partition mode for ordinary channels
adc_preempt_part_mode_enable	Enable partition mode for preempted channels
adc_preempt_auto_mode_enable	Enable auto conversion of preempted group at the end of ordinary conversion
adc_tempersensor_vintrv_enable	Enable internal temperature sensor and V <sub>INTRV</sub>
adc_ordinary_software_trigger_enable	Software trigger ordinary group conversion
adc_ordinary_software_trigger_status_get	Get the status of ordinary group conversion triggered by software
adc_preempt_software_trigger_enable	Software trigger preempted group conversion
adc_preempt_software_trigger_status_get	Get the status of preempted group conversion triggered by software
adc_ordinary_conversion_data_get	Get data of ordinary group conversion in non-master-slave mode
adc_combine_ordinary_conversion_data_get	Get the converted data of ordinary group in master/slave combined mode
adc_preempt_conversion_data_get	Get the converted data of preempted group

Function name	Description
adc_flag_get	Get the status of flag bits
adc_flag_clear	Clear flag bits

### 5.1.1 adc\_reset function

The table below describes the function adc\_reset.

Table 7. adc\_reset function

Name	Description
Function name	adc_reset
Function prototype	void adc_reset(adc_type *adc_x)
Function description	Reset all ADC registers to their reset values
Input parameter	adc_x: indicates the selected ADC This parameter is used to select one of ADC peripherals: ADC1, ADC2, ADC3
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset()

**Example:**

```
/* deinitialize adc1 */
adc_reset(ADC1);
```

### 5.1.2 adc\_enable function

The table below describes the function adc\_enable.

Table 8. adc\_enable function

Name	Description
Function name	adc_enable
Function prototype	void adc_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable/disable A/D converter
Input parameter 1	adc_x: indicates the selected ADC This parameter is used to select one of ADC peripherals: ADC1, ADC2, ADC3
Input parameter 2	new_state: indicates the pre-configured status of A/D converter This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable adc1 */
adc_enable(ADC1, TRUE);
```

*Note: Calling the function adc\_enable while the ADC is enabled triggers the conversion of ordinary channels.*

### 5.1.3 adc\_combine\_mode\_select function

The table below describes the function adc\_combine\_mode\_select.

**Table 9. adc\_combine\_mode\_select function**

Name	Description
Function name	adc_combine_mode_select
Function prototype	void adc_combine_mode_select(adc_combine_mode_type combine_mode)
Function description	Select master/slave combined mode for ADC1
Input parameter	combine_mode: master/slave combined mode for ADC1 This parameter can be any enumerated value in the adc_combine_mode_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### combine\_mode

The combine\_mode is used to select master/slave mode, including:

ADC\_INDEPENDENT\_MODE: Independent mode

ADC\_ORDINARY\_SMLT\_PREEMPT\_SMLT\_MODE:

Ordinary group simultaneous mode + preempted group simultaneous mode

ADC\_ORDINARY\_SMLT\_PREEMPT\_INTERLTRIG\_MODE:

Ordinary group simultaneous mode + preempted group alternate trigger mode

ADC\_ORDINARY\_SHORTSHIFT\_PREEMPT\_SMLT\_MODE:

Preempted group simultaneous mode + ordinary group short shift mode

ADC\_ORDINARY\_LONGSHIFT\_PREEMPT\_SMLT\_MODE:

Preempted group simultaneous mode + ordinary group long shift mode

ADC\_PREEMPT\_SMLT\_ONLY\_MODE: Preempted group simultaneous mode

ADC\_ORDINARY\_SMLT\_ONLY\_MODE: Ordinary group simultaneous mode

ADC\_ORDINARY\_SHORTSHIFT\_ONLY\_MODE: Ordinary group short shift mode

ADC\_ORDINARY\_LONGSHIFT\_ONLY\_MODE: Ordinary group long shift mode

ADC\_PREEMPT\_INTERLTRIG\_ONLY\_MODE: Preempted group alternate trigger mode

#### Example:

```
/* select combine mode as independent mode */  
adc_combine_mode_select(ADC_INDEPENDENT_MODE);
```

Note: adc\_combine\_mode\_select applies to ADC1 only, no effect on ADC2 and ADC3.

## 5.1.4 adc\_base\_default\_para\_init function

The table below describes the function adc\_base\_default\_para\_init.

**Table 10. adc\_base\_default\_para\_init function**

Name	Description
Function name	adc_base_default_para_init
Function prototype	void adc_base_default_para_init(adc_base_config_type *adc_base_struct)
Function description	Set the initial default value for the adc_base_struct
Input parameter	adc_base_struct: adc_base_config_type structure pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The initial default values of members of the adc\_base\_struct are as follows:

sequence_mode:	FALSE
repeat_mode:	FALSE
data_align:	ADC_RIGHT_ALIGNMENT
ordinary_channel_length:	1

**Example:**

```
/* initialize a adc_base_config_type structure */
adc_base_config_type adc_base_struct;
adc_base_default_para_init(&adc_base_struct);
```

## 5.1.5 adc\_base\_config function

The table below describes the function adc\_base\_config.

**Table 11. adc\_base\_config function**

Name	Description
Function name	adc_base_config
Function prototype	void adc_base_config(adc_type *adc_x, adc_base_config_type *adc_base_struct);
Function description	Initialize ADC registers with the specified parameters in the adc_base_struct
Input parameter 1	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Input parameter 2	adc_base_struct: adc_base_config_type structure pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**adc\_base\_config\_type structure**

The adc\_base\_config\_type is defined in the at32f403\_adc.h:

```
typedef struct
{
    confirm_state sequence_mode;
    confirm_state repeat_mode;
```

```

    adc_data_align_type      data_align;
    uint8_t                  ordinary_channel_length;
} adc_base_config_type; the member parameters are described as follows

sequence_mode
Set ADC sequence mode.
FALSE: Select a single channel for conversion
TRUE: Select multiple channels for conversion

repeat_mode
Set ADC repeat mode.
FALSE: when SQEN=0, trigger a single channel conversion each time; when SQEN=1, trigger the
       conversion of a group of channels each time
TRUE: when SQEN =0, repeatedly convert a single channel at each trigger; when SQEN=1,
       repeatedly convert a group of channels at each trigger until the ADCEN bit is cleared.

```

**data\_align**

Set data alignment of ADC

ADC\_RIGHT\_ALIGNMENT: right-aligned

ADC\_LEFT\_ALIGNMENT: left-aligned

**ordinary\_channel\_length**

Set the length of ordinary group ADC conversion

**Example:**

```

adc_base_config_type adc_base_struct;
adc_base_struct.sequence_mode = TRUE;
adc_base_struct.repeat_mode = FALSE;
adc_base_struct.data_align = ADC_RIGHT_ALIGNMENT;
adc_base_struct.ordinary_channel_length = 3;
adc_base_config(ADC1, &adc_base_struct);

```

## 5.1.6 adc\_dma\_mode\_enable function

The table below describes the function adc\_dma\_mode\_enable.

**Table 12. adc\_dma\_mode\_enable function**

Name	Description
Function name	adc_dma_mode_enable
Function prototype	void adc_dma_mode_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable DMA transfer for ordinary group conversion
Input parameter 1	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Input parameter 2	new_state: pre-configured status of ordinary group in DMA transfer mode This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```

/* enable dma transfer adc ordinary conversion data */
adc_dma_mode_enable(ADC1, TRUE);

```

Note: adc\_combine\_mode\_select applies to ADC1 and ADC3 only, no effect on ADC2

## 5.1.7 adc\_interrupt\_enable function

The table below describes the function adc\_interrupt\_enable.

**Table 13. adc\_interrupt\_enable function**

Name	Description
Function name	adc_interrupt_enable
Function prototype	void adc_interrupt_enable(adc_type *adc_x, uint32_t adc_int, confirm_state new_state)
Function description	Enable the selected ADC event interrupt
Input parameter 1	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Input parameter 2	adc_int: ADC event interrupt selection This parameter is used to select any event interrupt supported by ADC
Input parameter 3	new_state: indicates the pre-configured status of ADC event interrupts This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### adc\_int

The adc\_int is used to select and set event interrupts, with the following parameters:

ADC\_CCE\_INT: Interrupt enabled at the end of channel conversion

ADC\_VMOR\_INT: Interrupt enabled when voltage monitor is outside a threshold

ADC\_PCCE\_INT: Interrupt enabled at the end of preempted group conversion

#### Example:

```
/* enable voltage monitoring out of range interrupt */
adc_interrupt_enable(ADC1, ADC_VMOR_INT, TRUE);
```

## 5.1.8 adc\_calibration\_init function

The table below describes the function adc\_calibration\_init.

**Table 14. adc\_calibration\_init function**

Name	Description
Function name	adc_calibration_init
Function prototype	void adc_calibration_init(adc_type *adc_x)
Function description	Initialize calibration
Input parameter	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### Example:

```
/* initialize A/D calibration */
adc_calibration_init(ADC1);
```

## 5.1.9 adc\_calibration\_init\_status\_get function

The table below describes the function adc\_calibration\_init\_status\_get.

**Table 15. adc\_calibration\_init\_status\_get function**

Name	Description
Function name	adc_calibration_init_status_get
Function prototype	flag_status adc_calibration_init_status_get(adc_type *adc_x)
Function description	Get the status of initialization calibration
Input parameter	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Output parameter	NA
Return value	flag_status: indicates the status of calibration initialization Return SET or RESET.
Required preconditions	NA
Called functions	NA

**Example:**

```
/* wait initialize A/D calibration success */  
while(adc_calibration_init_status_get(ADC1));
```

## 5.1.10 adc\_calibration\_start function

The table below describes the function adc\_calibration\_start.

**Table 16. adc\_calibration\_start function**

Name	Description
Function name	adc_calibration_start
Function prototype	void adc_calibration_start(adc_type *adc_x)
Function description	Start calibration
Input parameter	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* start calibration process */  
adc_calibration_start(ADC1);
```

### 5.1.11 adc\_calibration\_status\_get function

The table below describes the function adc\_calibration\_status\_get.

**Table 17. adc\_calibration\_status\_get function**

Name	Description
Function name	adc_calibration_status_get
Function prototype	flag_status adc_calibration_status_get(adc_type *adc_x)
Function description	Get the status of calibration
Input parameter	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Output parameter	NA
Return value	flag_status: indicates the status of calibration Return SET or RESET
Required preconditions	NA
Called functions	NA

**Example:**

```
/* wait calibration success */
while(adc_calibration_status_get(ADC1));
```

### 5.1.12 adc\_voltage\_monitor\_enable function

The table below describes the function adc\_voltage\_monitor\_enable.

**Table 18. adc\_voltage\_monitor\_enable function**

Name	Description
Function name	adc_voltage_monitor_enable
Function prototype	void adc_voltage_monitor_enable(adc_type *adc_x, adc_voltage_monitoring_type adc_voltage_monitoring)
Function description	Enable voltage monitor for ordinary/preempted group and a single channel
Input parameter 1	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Input parameter 2	adc_voltage_monitoring: select ordinary group, preempted group or a single channel for voltage monitoring This parameter can be any enumerated value in the adc_voltage_monitoring_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### adc\_voltage\_monitoring

adc\_voltage\_monitoring is used to select one or more channels of ordinary group/preempted group for voltage monitoring, including:

ADC\_VMONITOR\_SINGLE\_ORDINARY:

Select a single ordinary channel for voltage monitoring

ADC\_VMONITOR\_SINGLE\_PREEMPT:

Select a single preempted channel for voltage monitoring

ADC\_VMONITOR\_SINGLE\_ORDINARY\_PREEMPT:

Select a single channel from ordinary or preempted group for voltage monitoring

ADC\_VMONITOR\_ALL\_ORDINARY:

Select all ordinary channels for voltage monitoring

ADC\_VMONITOR\_ALL\_PREEMPT:

Select all preempted channels for voltage monitoring

ADC\_VMONITOR\_ALL\_ORDINARY\_PREEMPT:

Select all ordinary and preempted channels for voltage monitoring

ADC\_VMONITOR\_NONE:

No channels need voltage monitoring

**Example:**

```
/* enable the voltage monitoring on all ordinary and preempt channels */
adc_voltage_monitor_enable(ADC1, ADC_VMONITOR_ALL_ORDINARY_PREEMPT);
```

### 5.1.13 adc\_voltage\_monitor\_threshold\_value\_set function

The table below describes the function adc\_voltage\_monitor\_threshold\_value\_set.

**Table 19. adc\_voltage\_monitor\_threshold\_value\_set function**

Name	Description
Function name	adc_voltage_monitor_threshold_value_set
Function prototype	void adc_voltage_monitor_threshold_value_set(adc_type *adc_x, uint16_t adc_high_threshold, uint16_t adc_low_threshold)
Function description	Configure the threshold of voltage monitoring
Input parameter 1	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Input parameter 2	adc_high_threshold: indicates the upper limit for voltage monitoring This parameter can be any value between 0x000 and 0xFFFF
Input parameter 3	adc_low_threshold: indicates the lower limit for voltage monitoring This parameter can be any value lower than that of the adc_high_threshold in the range of 0x000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* set voltage monitoring's high and low thresholds value */
adc_voltage_monitor_threshold_value_set(ADC1, 0xBBB, 0xAAA);
```

## 5.1.14 adc\_voltage\_monitor\_single\_channel\_select function

The table below describes the function adc\_voltage\_monitor\_single\_channel\_select.

**Table 20. adc\_voltage\_monitor\_single\_channel\_select function**

Name	Description
Function name	adc_voltage_monitor_single_channel_select
Function prototype	void adc_voltage_monitor_single_channel_select(adc_type *adc_x, adc_channel_select_type adc_channel)
Function description	Select a single channel for voltage monitoring
Input parameter 1	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Input parameter 2	adc_channel: select a single channel for voltage monitoring Refer to <a href="#">adc_channel</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### adc\_channel

The adc\_channel is used to select a single channel for voltage monitoring, including:

ADC\_CHANNEL\_0: ADC channel 0

ADC\_CHANNEL\_1: ADC channel 1

.....

ADC\_CHANNEL\_16: ADC channel 16

ADC\_CHANNEL\_17: ADC channel 17

### Example:

```
/* select the voltage monitoring's channel */
adc_voltage_monitor_single_channel_select(ADC1, ADC_CHANNEL_5);
```

## 5.1.15 adc\_ordinary\_channel\_set function

The table below describes the function adc\_ordinary\_channel\_set.

**Table 21. adc\_ordinary\_channel\_set function**

Name	Description
Function name	adc_ordinary_channel_set
Function prototype	void adc_ordinary_channel_set(adc_type *adc_x, adc_channel_select_type adc_channel, uint8_t adc_sequence, adc_sampletime_select_type adc_sampletime)
Function description	Configure ordinary channels, including such parameters as channel selection, conversion sequence number and sampling time
Input parameter 1	adc_x: indicates the selected ADC peripheral This parameter can be ADC1, ADC2, ADC3
Input parameter 2	adc_channel: indicates the selected channel Refer to <a href="#">adc_channel</a> for details.
Input parameter 3	adc_sequence: defines the sequence of channel conversion This parameter can be any value from 1 to 16.
Input parameter 4	adc_sampletime: defines the sampling time for channels Refer to <a href="#">adc_sampletime</a> for details

Name	Description
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**adc\_sampletime**

adc\_sampletime is used to configure the sampling time of channels, including:

ADC\_SAMPLETIME\_1\_5: sampling time is 1.5 ADCCLK cycles

ADC\_SAMPLETIME\_7\_5: sampling time is 7.5 ADCCLK cycles

ADC\_SAMPLETIME\_13\_5: sampling time is 13.5 ADCCLK cycles

ADC\_SAMPLETIME\_28\_5: sampling time is 28.5 ADCCLK cycles

ADC\_SAMPLETIME\_41\_5: sampling time is 41.5 ADCCLK cycles

ADC\_SAMPLETIME\_55\_5: sampling time is 55.5 ADCCLK cycles

ADC\_SAMPLETIME\_71\_5: sampling time is 71.5 ADCCLK cycles

ADC\_SAMPLETIME\_239\_5: sampling time is 239.5 ADCCLK cycles

**Example:**

```
/* set ordinary channel's corresponding rank in the sequencer and sample time */
adc_ordinary_channel_set(ADC1, ADC_CHANNEL_4, 1, ADC_SAMPLETIME_239_5);
adc_ordinary_channel_set(ADC1, ADC_CHANNEL_5, 2, ADC_SAMPLETIME_239_5);
```

### 5.1.16 adc\_preempt\_channel\_length\_set function

The table below describes the function adc\_preempt\_channel\_length\_set.

**Table 22. adc\_preempt\_channel\_length\_set function**

Name	Description
Function name	adc_preempt_channel_length_set
Function prototype	void adc_preempt_channel_length_set(adc_type *adc_x, uint8_t adc_channel_length)
Function description	set the length of preempted channel conversion
Input parameter 1	adc_x: indicates the selected ADC This parameter can be any ADC1, ADC2, ADC3
Input parameter 2	adc_channel_length: set the length of preempted channel conversion This parameter can be any value from 0x1 to 0x4.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* set preempt channel length */
adc_preempt_channel_length_set(ADC1, 3);
```

### 5.1.17 adc\_preempt\_channel\_set function

The table below describes the function adc\_preempt\_channel\_set.

**Table 23. adc\_preempt\_channel\_set function**

Name	Description
Function name	adc_preempt_channel_set
Function prototype	void adc_preempt_channel_set(adc_type *adc_x, adc_channel_select_type adc_channel, uint8_t adc_sequence, adc_sampletime_select_type adc_sampletime)
Function description	Configure preempted group, including such parameters as channel selection, conversion sequence number and sampling time
Input parameter 1	adc_x: indicates the selected ADC This parameter can be any ADC1, ADC2, ADC3
Input parameter 2	adc_channel: indicates the selected channel Refer to <a href="#">adc_channel</a> for details
Input parameter 3	adc_sequence: set the sequence number for channel conversion This parameter can be any value from 1 to 4
Input parameter 4	adc_sampletime: set the sampling time for channels Refer to <a href="#">adc_sampletime</a> for details
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* set ordinary channel's corresponding rank in the sequencer and sample time */
adc_preempt_channel_set(ADC1, ADC_CHANNEL_7, 1, ADC_SAMPLETIME_239_5);
adc_preempt_channel_set(ADC1, ADC_CHANNEL_8, 2, ADC_SAMPLETIME_239_5);
```

### 5.1.18 adc\_ordinary\_conversion\_trigger\_set function

The table below describes the function adc\_ordinary\_conversion\_trigger\_set.

**Table 24. adc\_ordinary\_conversion\_trigger\_set function**

Name	Description
Function name	adc_ordinary_conversion_trigger_set
Function prototype	void adc_ordinary_conversion_trigger_set(adc_type *adc_x, adc_ordinary_trig_select_type adc_ordinary_trig, confirm_state new_state)
Function description	Enable trigger mode and select trigger events for ordinary group conversion
Input parameter 1	adc_x: indicates the selected ADC This parameter can be any ADC1, ADC2, ADC3
Input parameter 2	adc_ordinary_trig: indicates the selected trigger event for ordinary group This parameter can be any enumerated value in the adc_ordinary_trig_select_type
Input parameter 3	new_state: indicates the pre-configured status of trigger mode This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA

Name	Description
Called functions	NA

**adc\_ordinary\_trig**

adc\_ordinary\_trig is used to select a trigger event for ordinary group conversion, including:

**Trigger events for ADC1 &ADC2:**

ADC12_ORDINARY_TRIG_TMR1CH1:	TMR1 CH1 event
ADC12_ORDINARY_TRIG_TMR1CH2:	TMR1 CH2 event
ADC12_ORDINARY_TRIG_TMR1CH3:	TMR1 CH3 event
ADC12_ORDINARY_TRIG_TMR2CH2:	TMR2 CH2 event
ADC12_ORDINARY_TRIG_TMR3TRGOUT:	TMR3 TRGOUT event
ADC12_ORDINARY_TRIG_TMR4CH4:	TMR4 CH4 event
ADC12_ORDINARY_TRIG_EXINT11_TMR8TRGOUT:	EXINT11/TMR8 TRGOUT event
ADC12_ORDINARY_TRIG_SOFTWARE:	Software-triggered event
ADC12_ORDINARY_TRIG_TMR1TRGOUT:	TMR1 TRGOUT event
ADC12_ORDINARY_TRIG_TMR8CH1:	TMR8 CH1 event
ADC12_ORDINARY_TRIG_TMR8CH2:	TMR8 CH2 event

**Trigger events for ADC3:**

ADC3_ORDINARY_TRIG_TMR3CH1:	TMR3 CH1 event
ADC3_ORDINARY_TRIG_TMR2CH3:	TMR2 CH3 event
ADC3_ORDINARY_TRIG_TMR1CH3:	TMR1 CH3 event
ADC3_ORDINARY_TRIG_TMR8CH1:	TMR8 CH1 event
ADC3_ORDINARY_TRIG_TMR8TRGOUT:	TMR8 TRGOUT event
ADC3_ORDINARY_TRIG_TMR5CH1:	TMR5 CH1 event
ADC3_ORDINARY_TRIG_TMR5CH3:	TMR5 CH3 event
ADC3_ORDINARY_TRIG_SOFTWARE:	Software-triggered event
ADC3_ORDINARY_TRIG_TMR1TRGOUT:	TMR1 TRGOUT event
ADC3_ORDINARY_TRIG_TMR1CH1:	TMR1 CH1 event
ADC3_ORDINARY_TRIG_TMR8CH3:	TMR8 CH3 event

**Example:**

```
/* set ordinary external trigger event */
adc_ordinary_conversion_trigger_set(ADC1, ADC12_ORDINARY_TRIG_TMR1CH1, TRUE);
```

## 5.1.19 adc\_preempt\_conversion\_trigger\_set function

The table below describes the function adc\_preempt\_conversion\_trigger\_set.

**Table 25. adc\_preempt\_conversion\_trigger\_set function**

Name	Description
Function name	adc_preempt_conversion_trigger_set
Function prototype	void adc_preempt_conversion_trigger_set(adc_type *adc_x, adc_preempt_trig_select_type adc_preempt_trig, confirm_state new_state)
Function description	Enable trigger mode and trigger events for preempted group conversion
Input parameter 1	adc_x: indicates the selected ADC This parameter can be any ADC1, ADC2, ADC3
Input parameter 2	adc_preempt_trig: indicates the selected trigger event for preempted group This parameter can be any enumerated value in the adc_preempt_trig_select_type
Input parameter 3	new_state: indicates the pre-configured status of trigger mode This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### adc\_preempt\_trig

adc\_preempt\_trig is used to select a trigger event for preempted group conversion, including:

#### Trigger events for ADC1 &ADC2:

ADC12_PREEMPT_TRIG_TMR1TRGOUT:	TMR1 TRGOUT event
ADC12_PREEMPT_TRIG_TMR1CH4:	TMR1 CH4 event
ADC12_PREEMPT_TRIG_TMR2TRGOUT:	TMR2 TRGOUT event
ADC12_PREEMPT_TRIG_TMR2CH1:	TMR2 CH1 event
ADC12_PREEMPT_TRIG_TMR3CH4:	TMR3 CH4 event
ADC12_PREEMPT_TRIG_TMR4TRGOUT:	TMR4 TRGOUT event
ADC12_PREEMPT_TRIG_EXINT15_TMR8CH4:	EXINT 15/TMR8 CH4 event
ADC12_PREEMPT_TRIG_SOFTWARE:	Software-triggered event
ADC12_PREEMPT_TRIG_TMR1CH1:	TMR1 CH1 event
ADC12_PREEMPT_TRIG_TMR8CH1:	TMR8 CH1 event
ADC12_PREEMPT_TRIG_TMR8TRGOUT:	TMR8 TRGOUT event

#### Trigger events for ADC3:

ADC3_PREEMPT_TRIG_TMR1TRGOUT:	TMR3 TRGOUT event
ADC3_PREEMPT_TRIG_TMR1CH4:	TMR1 CH4 event
ADC3_PREEMPT_TRIG_TMR4CH3:	TMR4 CH3 event
ADC3_PREEMPT_TRIG_TMR8CH2:	TMR8 CH2 event
ADC3_PREEMPT_TRIG_TMR8CH4:	TMR8 CH4 event
ADC3_PREEMPT_TRIG_TMR5TRGOUT:	TMR5 TRGOUT event
ADC3_PREEMPT_TRIG_TMR5CH4:	TMR5 CH4 event
ADC3_PREEMPT_TRIG_SOFTWARE:	Software-triggered event
ADC3_PREEMPT_TRIG_TMR1CH1:	TMR1 CH1 event
ADC3_PREEMPT_TRIG_TMR1CH2:	TMR1 CH2 event
ADC3_PREEMPT_TRIG_TMR8TRGOUT:	TMR8 TRGOUT event

**Example:**

```
/* set preempt external trigger event */
adc_preempt_conversion_trigger_set(ADC1, ADC12_PREEMPT_TRIG_SOFTWARE, TRUE);
```

### 5.1.20 adc\_preempt\_offset\_value\_set function

The table below describes the function adc\_preempt\_offset\_value\_set.

**Table 26. adc\_preempt\_offset\_value\_set function**

Name	Description
Function name	adc_preempt_offset_value_set
Function prototype	void adc_preempt_offset_value_set(adc_type *adc_x, adc_preempt_channel_type adc_preempt_channel, uint16_t adc_offset_value)
Function description	set the offset value of preempted group conversion
Input parameter 1	adc_x: selected ADC This parameter can be ADC1, ADC2, ADC3
Input parameter 2	adc_preempt_channel: indicates the selected channel Refer to <a href="#">adc_preempt_channel</a> for details
Input parameter 3	adc_offset_value: set the offset value for the selected channel This parameter can be any value from 0x000 to 0xFFFF.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**adc\_preempt\_channel**

adc\_preempt\_channel is used to set an offset value for the selected channel, including:

ADC_PREEMPT_CHANNEL_1:	Preempted channel 1
ADC_PREEMPT_CHANNEL_2:	Preempted channel 2
ADC_PREEMPT_CHANNEL_3:	Preempted channel 3
ADC_PREEMPT_CHANNEL_4:	Preempted channel 4

**Example:**

```
/* set preempt channel's conversion value offset */
adc_preempt_offset_value_set(ADC1, ADC_PREEMPT_CHANNEL_1, 0x111);
adc_preempt_offset_value_set(ADC1, ADC_PREEMPT_CHANNEL_2, 0x222);
adc_preempt_offset_value_set(ADC1, ADC_PREEMPT_CHANNEL_3, 0x333);
```

## 5.1.21 adc\_ordinary\_part\_count\_set function

The table below describes the function adc\_ordinary\_part\_count\_set.

**Table 27. adc\_ordinary\_part\_count\_set function**

Name	Description
Function name	adc_ordinary_part_count_set
Function prototype	void adc_ordinary_part_count_set(adc_type *adc_x, uint8_t adc_channel_count)
Function description	Set the number of ordinary channels at each triggered conversion in partition mode
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3
Input parameter 2	adc_channel_count: indicates the number of ordinary group in partition mode This parameter can be any value from 0x1 to 0x8
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* set partitioned mode channel count */
adc_ordinary_part_count_set(ADC1, 2);
```

*Note: In partition mode, only the number of ordinary group is settable, and that of preempted group is fixed 1.*

## 5.1.22 adc\_ordinary\_part\_mode\_enable function

The table below describes the function adc\_ordinary\_part\_mode\_enable.

**Table 28. adc\_ordinary\_part\_mode\_enable function**

Name	Description
Function name	adc_ordinary_part_mode_enable
Function prototype	void adc_ordinary_part_mode_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable partition mode for ordinary channels
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3
Input parameter 2	new_state: indicates the pre-configured status for partition mode of ordinary channels This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable the partitioned mode on ordinary channel */
adc_ordinary_part_mode_enable(ADC1, TRUE);
```

## 5.1.23 adc\_preempt\_part\_mode\_enable function

The table below describes the function adc\_preempt\_part\_mode\_enable.

**Table 29. adc\_preempt\_part\_mode\_enable**

Name	Description
Function name	adc_preempt_part_mode_enable
Function prototype	void adc_preempt_part_mode_enable(adc_type *adc_x, confirm_state new_state)
Function description	Enable partition mode for preempted channels
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3.
Input parameter 2	new_state: indicates the pre-configured status for partition mode of preempted channels This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable the partitioned mode on preempt channel */
adc_preempt_part_mode_enable(ADC1, TRUE);
```

## 5.1.24 adc\_preempt\_auto\_mode\_enable function

The table below describes the function adc\_preempt\_auto\_mode\_enable.

**Table 30. adc\_preempt\_auto\_mode\_enable function**

Name	Description
Function name	adc_preempt_auto_mode_enable
Function prototype	void adc_preempt_auto_mode_enable(adc_type *adc_x, confirm_state,new_state)
Function description	Enable auto preempted group conversion at the end of ordinary group conversion
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3
Input parameter 2	new_state: indicates the pre-configured status for auto preempted group conversion This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable automatic preempt group conversion */
adc_preempt_auto_mode_enable(ADC1, TRUE);
```

## 5.1.25 adc\_tempersensor\_vinrv\_enable function

The table below describes the function adc\_tempersensor\_vinrv\_enable.

**Table 31. adc\_tempersensor\_vinrv\_enable function**

Name	Description
Function name	adc_tempersensor_vinrv_enable
Function prototype	void adc_tempersensor_vinrv_enable(confirm_state new_state)
Function description	Enable internal temperature sensor and V <sub>INTRV</sub>
Input parameter	new_state: indicates the pre-configured status for internal temperature sensor and V <sub>INTRV</sub> This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable the temperature sensor and vinrv channel */
adc_tempersensor_vinrv_enable(TRUE);
```

## 5.1.26 adc\_ordinary\_software\_trigger\_enable function

The table below describes the function adc\_ordinary\_software\_trigger\_enable.

**Table 32. adc\_ordinary\_software\_trigger\_enable function**

Name	Description
Function name	adc_ordinary_software_trigger_enable
Function prototype	void adc_ordinary_software_trigger_enable(adc_type *adc_x, confirm_state new_state)
Function description	Trigger ordinary group conversion by software
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3
Input parameter 2	new_state: indicates the pre-configured status for software-triggered ordinary group conversion This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable ordinary software start conversion */
adc_ordinary_software_trigger_enable(ADC1, TRUE);
```

## 5.1.27 adc\_ordinary\_software\_trigger\_status\_get function

The table below describes the function adc\_ordinary\_software\_trigger\_status\_get.

**Table 33. adc\_ordinary\_software\_trigger\_status\_get function**

Name	Description
Function name	adc_ordinary_software_trigger_status_get
Function prototype	flag_status adc_ordinary_software_trigger_status_get(adc_type *adc_x)
Function description	Get the status of software-triggered ordinary group conversion
Input parameter	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3
Output parameter	NA
Return value	flag_status: indicates the status of software-triggered ordinary group conversion This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

**Example:**

```
/* wait ordinary software start conversion */
while(adc_ordinary_software_trigger_status_get(ADC1));
```

## 5.1.28 adc\_preempt\_software\_trigger\_enable function

The table below describes the function adc\_preempt\_software\_trigger\_enable.

**Table 34. adc\_preempt\_software\_trigger\_enable function**

Name	Description
Function name	adc_preempt_software_trigger_enable
Function prototype	void adc_preempt_software_trigger_enable(adc_type *adc_x, confirm_state new_state)
Function description	Preempted group conversion triggered by software event
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3.
Input parameter 2	new_state: indicates the pre-configured status of software-triggered preempted group conversion This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable preempt software start conversion */
adc_preempt_software_trigger_enable(ADC1, TRUE);
```

## 5.1.29 adc\_preempt\_software\_trigger\_status\_get function

The table below describes the function adc\_preempt\_software\_trigger\_status\_get.

**Table 35. adc\_preempt\_software\_trigger\_status\_get function**

Name	Description
Function name	adc_preempt_software_trigger_status_get
Function prototype	flag_status adc_preempt_software_trigger_status_get(adc_type *adc_x)
Function description	Get the status of software-triggered preempted group conversion
Input parameter	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3
Output parameter	NA
Return value	flag_status: indicates the status of software-triggered preempted group conversion This parameter can be SET or RESET
Required preconditions	NA
Called functions	NA

**Example:**

```
/* wait preempt software start conversion */
while(adc_preempt_software_trigger_status_get(ADC1));
```

## 5.1.30 adc\_ordinary\_conversion\_data\_get function

The table below describes the function adc\_ordinary\_conversion\_data\_get.

**Table 36. adc\_ordinary\_conversion\_data\_get function**

Name	Description
Function name	adc_ordinary_conversion_data_get
Function prototype	uint16_t adc_ordinary_conversion_data_get(adc_type *adc_x)
Function description	Get the converted data of ordinary group in non-master/slave mode
Input parameter	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3
Output parameter	NA
Return value	16-bit converted data by ordinary group
Required preconditions	NA
Called functions	NA

**Example:**

```
uint16_t adc1_ordinary_index = 0;
adc1_ordinary_index = adc_ordinary_conversion_data_get(ADC1);
```

*Note: This function can be used only when the ADC is configured with a single channel.*

### 5.1.31 adc\_combine\_ordinary\_conversion\_data\_get function

The table below describes the function adc\_combine\_ordinary\_conversion\_data\_get

**Table 37. adc\_combine\_ordinary\_conversion\_data\_get function**

Name	Description
Function name	adc_combine_ordinary_conversion_data_get
Function prototype	uint32_t adc_combine_ordinary_conversion_data_get(void)
Function description	Get the converted data of ordinary group in master/slave combine mode
Input parameter	NA
Output parameter	NA
Return value	32-bit converted data of ordinary group (upper 16 bits are for ADC2, the lower 16bits for ADC1)
Required preconditions	NA
Called functions	NA

**Example:**

```
uint32_t common_ordinary_index = 0;
common_ordinary_index = adc_combine_ordinary_conversion_data_get();
```

*Note: This function can be used only when the ADC is configured in master/slave combined mode and each of ADCs is configured with a single channel.*

### 5.1.32 adc\_preempt\_conversion\_data\_get function

The table below describes the function adc\_preempt\_conversion\_data\_get.

**Table 38. adc\_preempt\_conversion\_data\_get function**

Name	Description
Function name	adc_preempt_conversion_data_get
Function prototype	uint16_t adc_preempt_conversion_data_get(adc_type *adc_x, adc_preempt_channel_type adc_preempt_channel)
Function description	Get the converted data of preempted group
Input parameter 1	adc_x: selected ADC This parameter can be ADC1, ADC2, ADC3
Input parameter 2	adc_preempt_channel: the selected preempted channel Refer to <a href="#">adc_preempt_channel</a> for details
Output parameter	NA
Return value	16-bit converted data by preempted group
Required preconditions	NA
Called functions	NA

**Example:**

```
uint16_t adc1_preempt_valuetab[3] = {0};
adc1_preempt_valuetab[0] = adc_preempt_conversion_data_get(ADC1, ADC_PREEMPT_CHANNEL_1);
adc1_preempt_valuetab[1] = adc_preempt_conversion_data_get(ADC1, ADC_PREEMPT_CHANNEL_2);
adc1_preempt_valuetab[2] = adc_preempt_conversion_data_get(ADC1, ADC_PREEMPT_CHANNEL_3);
```

### 5.1.33 adc\_flag\_get function

The table below describes the function adc\_flag\_get.

**Table 39. adc\_flag\_get function**

Name	Description
Function name	adc_flag_get
Function prototype	flag_status adc_flag_get(adc_type *adc_x, uint8_t adc_flag)
Function description	Get the status of the flag bit
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3.
Input parameter 2	adc_flag: indicates the selected flag Refer to <a href="#">adc_flag</a> for details.
Output parameter	NA
Return value	flag_status: the status for the selected flag bit Return value can be SET or RESET.
Required preconditions	NA
Called functions	NA

#### adc\_flag

The adc\_flag is used to select a flag to get its status, including:

- ADC\_VMOR\_FLAG: Voltage monitor outside threshold
- ADC\_CCE\_FLAG: End of channel conversion
- ADC\_PCCE\_FLAG: End of preempted group conversion
- ADC\_PCCS\_FLAG: Start of preempted group conversion
- ADC\_OCCS\_FLAG: Start of ordinary group conversion

#### Example:

```
/* check if wakeup preempted channelsconversion end flag is set */
if(adc_flag_get(ADC1, ADC_PCCE_FLAG) != RESET)
```

### 5.1.34 adc\_interrupt\_flag\_get function

The table below describes the function adc\_interrupt\_flag\_get.

**Table 40. adc\_interrupt\_flag\_get function**

Name	Description
Function name	adc_interrupt_flag_get
Function prototype	flag_status adc_interrupt_flag_get(adc_type *adc_x, uint8_t adc_flag)
Function description	Get the status of the interrupt flag bit
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3.
Input parameter 2	adc_flag: indicates the selected flag Refer to <a href="#">adc_flag</a> for details.
Output parameter	NA
Return value	flag_status: the status for the selected flag bit Return value can be SET or RESET.
Required preconditions	NA
Called functions	NA

**adc\_flag**

The adc\_flag is used to select a flag to get its status, including:

ADC\_VMOR\_FLAG: Voltage monitor outside threshold

ADC\_CCE\_FLAG: End of channel conversion

ADC\_PCCE\_FLAG: End of preempted group conversion

**Example**

```
/* check if wakeup preempted channelsconversion end flag is set */  
if(adc_interrupt_flag_get(ADC1, ADC_PCCE_FLAG) != RESET)
```

### 5.1.35 adc\_flag\_clear function

The table below describes the function adc\_flag\_clear.

**Table 41. adc\_flag\_clear function**

Name	Description
Function name	adc_flag_clear
Function prototype	void adc_flag_clear(adc_type *adc_x, uint32_t adc_flag)
Function description	Clear the flag bits that have been set.
Input parameter 1	adc_x: indicates the selected ADC This parameter can be ADC1, ADC2, ADC3.
Input parameter 2	adc_flag: select a flag to be cleared Refer to <a href="#">adc_flag</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* preempted channelsconversion end flag clear */  
adc_flag_clear(ADC1, ADC_PCCE_FLAG);
```

## 5.2 Battery powered domain (BPR)

The BPR register structure bpr\_type is defined in the “at32f403\_bpr.h”:

```
/*
 * @brief type define bpr register all
 */
typedef struct
{

} bpr_type;
```

The table below gives a list of the BPR registers.

**Table 42. Summary of BPR registers**

Register	Description
dt1	Battery powered data register 1
dt2	Battery powered data register 2
dt3	Battery powered data register 3
dt4	Battery powered data register 4
dt5	Battery powered data register 5
dt6	Battery powered data register 6
dt7	Battery powered data register 7
dt8	Battery powered data register 8
dt9	Battery powered data register 9
dt10	Battery powered data register 10
rtccal	RTC calibration register
ctrl	BPR control register
ctrlsts	BPR control/status register
dt11	Battery powered data register 11
dt12	Battery powered data register 12
dt13	Battery powered data register 13
dt14	Battery powered data register 14
dt15	Battery powered data register 15
dt16	Battery powered data register 16
dt17	Battery powered data register 17
dt18	Battery powered data register 18
dt19	Battery powered data register 19
dt20	Battery powered data register 20
dt21	Battery powered data register 21
dt22	Battery powered data register 22
dt23	Battery powered data register 23
dt24	Battery powered data register 24
dt25	Battery powered data register 25
dt26	Battery powered data register 26
dt27	Battery powered data register 27

Register	Description
dt28	Battery powered data register 28
dt29	Battery powered data register 29
dt30	Battery powered data register 30
dt31	Battery powered data register 31
dt32	Battery powered data register 32
dt33	Battery powered data register 33
dt34	Battery powered data register 34
dt35	Battery powered data register 35
dt36	Battery powered data register 36
dt37	Battery powered data register 37
dt38	Battery powered data register 38
dt39	Battery powered data register 39
dt40	Battery powered data register 40
dt41	Battery powered data register 41
dt42	Battery powered data register 42

The table below gives a list of BP library functions.

**Table 43. Summary of BP library functions**

Function name	Description
bpr_reset	Reset all BPR registers to their default values
bpr_flag_get	Get flag status
bpr_flag_clear	Clear flag status
bpr_interrupt_enable	Enable tamper detection interrupt
bpr_data_read	Read BPR data registers
bpr_data_write	Write BPR data registers
bpr_rtc_output_select	Select event output
bpr_rtc_clock_calibration_value_set	Set clock calibration
bpr_tamper_pin_enable	Enable tamper detection
bpr_tamper_pin_active_level_set	Select tamper detection active level

## 5.2.1 bpr\_reset function

The table below describes the function bpr\_reset.

**Table 44. bpr\_reset function**

Name	Description
Function name	bpr_reset
Function prototype	void bpr_reset(void);
Function description	Reset all BPR data registers to their default values
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void crm_batteryPoweredDomainReset(confirm_state new_state);

**Example:**

```
bpr_reset();
```

## 5.2.2 bpr\_flag\_get function

The table below describes the function bpr\_flag\_get.

**Table 45. bpr\_flag\_get function**

Name	Description
Function name	bpr_flag_get
Function prototype	flag_status bpr_flag_get(uint32_t flag);
Function description	Get flat status
Input parameter 1	Flag: flag selection Refer to the “flag” descriptions below for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

### flag

This is used for flag selection, including:

BPR\_TAMPER\_INTERRUPT\_FLAG: tamper detection interrupt

BPR\_TAMPER\_EVENT\_FLAG: tamper detection event

**Example:**

```
bpr_flag_get(BPR_TAMPER_INTERRUPT_FLAG);
```

### 5.2.3 bpr\_interrupt\_flag\_get function

The table below describes the function bpr\_interrupt\_flag\_get function.

**Table 46. bpr\_interrupt\_flag\_get function**

Name	Description
Function name	bpr_interrupt_flag_get
Function prototype	flag_status bpr_interrupt_flag_get(uint32_t flag);
Function description	Get flat status and judge the corresponding interrupt flag bit
Input parameter 1	flag: flag selection Refer to the “flag” descriptions below for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

#### flag

This is used for flag selection, including:

BPR\_TAMPER\_INTERRUPT\_FLAG: tamper detection interrupt  
BPR\_TAMPER\_EVENT\_FLAG: tamper detection event

#### Example

```
bpr_interrupt_flag_get(BPR_TAMPER_INTERRUPT_FLAG);
```

### 5.2.4 bpr\_flag\_clear function

The table below describes the function bpr\_flag\_clear.

**Table 47. bpr\_flag\_clear function**

Name	Description
Function name	bpr_flag_clear
Function prototype	void bpr_flag_clear(uint32_t flag);
Function description	Clear flag
Input parameter 1	Flag: flag selection Refer to the “flag” descriptions below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### flag

This is used for flag selection, including:

BPR\_TAMPER\_INTERRUPT\_FLAG: tamper detection interrupt  
BPR\_TAMPER\_EVENT\_FLAG: tamper detection event

#### Example:

```
bpr_flag_clear(BPR_TAMPER_INTERRUPT_FLAG);
```

## 5.2.5 bpr\_interrupt\_enable function

The table below describes the function bpr\_interrupt\_enable.

**Table 48. bpr\_interrupt\_enable function**

Name	Description
Function name	bpr_interrupt_enable
Function prototype	void bpr_interrupt_enable(confirm_state new_state);
Function description	Enable tamper detection interrupt
Input parameter 1	new_state: tamper detection interrupt status This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
bpr_interrupt_enable(TRUE);
```

## 5.2.6 bpr\_data\_read function

The table below describes the function bpr\_data\_read.

**Table 49. bpr\_data\_read function**

Name	Description
Function name	bpr_data_read
Function prototype	uint16_t bpr_data_read(bpr_data_type bpr_data);
Function description	Read BPR data registers
Input parameter 1	bpr_data: data register Refer to the “bpr_data” descriptions below for details.
Output parameter	NA
Return value	Return BPR data register value
Required preconditions	NA
Called functions	NA

### bpr\_data

Data registers.

BPR\_DATA1: Data register 1

BPR\_DATA2: Data register 2

BPR\_DATA41: Data register 41

BPR\_DATA42: Data register 42

**Example:**

```
bpr_data_read(BPR_DATA1);
```

## 5.2.7 bpr\_data\_write function

The table below describes the function bpr\_data\_write.

**Table 50. bpr\_data\_write function**

Name	Description
Function name	bpr_data_write
Function prototype	void bpr_data_write(bpr_data_type bpr_data, uint16_t data_value);
Function description	Write BPR data registers
Input parameter 1	bpr_data: data register Refer to the “bpr_data” descriptions below for details.
Input parameter 2	data_value: 16-bit data
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### bpr\_data

Data registers.

BPR\_DATA1: Data register 1

BPR\_DATA2: Data register 2

BPR\_DATA41: Data register 41

BPR\_DATA42: Data register 42

### Example:

```
bpr_data_write(BPR_DATA1, 0x5A5A);
```

## 5.2.8 bpr\_RTC\_output\_select function

The table below describes the function bpr\_RTC\_output\_select.

**Table 51. bpr\_RTC\_output\_select function**

Name	Description
Function name	bpr_RTC_output_select
Function prototype	void bpr_RTC_output_select(bpr_RTC_output_type output_source);
Function description	Select event output
Input parameter 1	output_source: output event Refer to the “output_source” descriptions below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### output\_source

Select an event output.

BPR\_RTC\_OUTPUT\_NONE: No output

BPR\_RTC\_OUTPUT\_CLOCK\_CAL\_BEFORE: divided-by-64 clock output before calibration

BPR\_RTC\_OUTPUT\_ALARM: Pulse alarm event output

BPR\_RTC\_OUTPUT\_SECOND: Pulse second event output

BPR\_RTC\_OUTPUT\_CLOCK\_CAL\_AFTER: divided-by-64 clock output after calibration

### Example:

```
bpr_RTC_output_select(BPR_RTC_OUTPUT_ALARM);
```

## 5.2.9 bpr\_RTC\_clock\_calibration\_value\_set function

The table below describes the function bpr\_RTC\_clock\_calibration\_value\_set.

**Table 52. bpr\_RTC\_clock\_calibration\_value\_set function**

Name	Description
Function name	bpr_RTC_clock_calibration_value_set
Function prototype	void bpr_RTC_clock_calibration_value_set(uint8_t calibration_value);
Function description	Set clock calibration
Input parameter 1	Value: calibration value ranges from 0 to 0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
bpr_RTC_clock_calibration_value_set(0x7F);
```

## 5.2.10 bpr\_tamper\_pin\_enable function

The table below describes the function bpr\_tamper\_pin\_enable.

**Table 53. bpr\_tamper\_pin\_enable function**

Name	Description
Function name	bpr_tamper_pin_enable
Function prototype	void bpr_tamper_pin_enable(confirm_state new_state);
Function description	Enable tamper detection
Input parameter 1	new_state: interrupt enable status This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
bpr_tamper_pin_enable(TRUE);
```

## 5.2.11 bpr\_tamper\_pin\_active\_level\_set function

The table below describes the function bpr\_tamper\_pin\_active\_level\_set.

**Table 54. bpr\_tamper\_pin\_active\_level\_set function**

Name	Description
Function name	bpr_tamper_pin_active_level_set
Function prototype	void bpr_tamper_pin_active_level_set(bpr_tamper_pin_active_level_type active_level);
Function description	Set tamper detection active level
Input parameter 1	active_level: tamper detection active level Refer to the "active_level" descriptions below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### active\_level

Select tamper detection active level.

BPR\_TAMPER\_PIN\_ACTIVE\_HIGH: High

BPR\_TAMPER\_PIN\_ACTIVE\_LOW: Low

### Example:

```
bpr_tamper_pin_active_level_set(BPR_TAMPER_PIN_ACTIVE_HIGH);
```

## 5.3 Controller area network (CAN)

CAN register structure can\_type is defined in the “at32f403\_can.h”:

```
/*
 * @brief type define can register all
 */
typedef struct
{
    ...
} can_type;
```

The table below gives a list of the CAN registers.

**Table 55. Summary of ADC registers**

Register	Description
mctrl	CAN master control register
msts	CAN master status register
tsts	CAN transmit status register
rf0	CAN receive FIFO 0 register
fr1	CAN receive FIFO 1 register
inten	CAN interrupt enable register
ests	CAN error status register
btmg	CAN bit timing register
tmi0	Transmit mailbox 0 identifier register
tmc0	Transmit mailbox 0 data length and time stamp register
tmdtl0	Transmit mailbox 0 data byte low register
tmdth0	Transmit mailbox 0 data byte high register
tmi1	Transmit mailbox 1 identifier register
tmc1	Transmit mailbox 1 data length and time stamp register
tmdtl1	Transmit mailbox 1 data byte low register
tmdth1	Transmit mailbox 1 data byte high register
tmi2	Transmit mailbox 2 identifier register
tmc2	Transmit mailbox 2 data length and time stamp register
tmdtl2	Transmit mailbox 2 data byte low register
tmdth2	Transmit mailbox 2 data byte high register
rfi0	Receive FIFO0 mailbox identifier register
rfc0	Receive FIFO0 mailbox data length and time stamp register
rfdtl0	Receive FIFO0 mailbox data byte low register
rfdth0	Receive FIFO0 mailbox data byte high register
rfi1	Receive FIFO1 mailbox identifier register
rfc1	Receive FIFO1 mailbox data length and time stamp register
rfdtl1	Receive FIFO1 mailbox data byte low register
rfdth1	Receive FIFO1 mailbox data byte high register
fctrl	CAN filter control register
fmcfg	CAN filter mode configuration register
fscfg	CAN filter size configuration register

Register	Description
frf	CAN filter FIFO assosication register
facfg	CAN filter activate control register
fb0f1	CAN filter bank 0 filter register 1
fb0f2	CAN filter bank 0 filter register 2
fb1f1	CAN filter bank 1 filter register 1
fb1f2	CAN filter bank 1 filter register 2
...	...
fb13f1	CAN filter bank 13 filter register 1
fb13f2	CAN filter bank 13 filter register 2

The table below gives a list of CAN library functions.

**Table 56. Summary of CAN library functions**

Function name	Description
can_reset	Reset all CAN registers to their reset values
can_baudrate_default_para_init	Configure the CAN baud rate initial structure with the initial value
can_baudrate_set	Configure CAN baud rate
can_default_para_init	Configure the CAN initial structure with the initial value
can_base_init	Initialize CAN registers with the specified parameters in the can_base_struct
can_filter_default_para_init	Configure the CAN filter initial structure with the initial value
can_filter_init	Initialize CAN registers with the specified parameters in the can_filter_init_struct
can_debug_transmission_prohibit	Select to 93isable/enable message reception and transmission when debug
can_ttc_mode_enable	Enable time-triggered mode
can_message_transmit	Transmit a frame of message
can_transmit_status_get	Get the status of transmission
can_transmit_cancel	Cancel transmission
can_message_receive	Receive a frame of message
can_receive_fifo_release	Release receive FIFO
can_receive_message_pending_get	Get the count of pending messages in FIFO
can_operating_mode_set	Configure CAN operating mode
can_doze_mode_enter	Enter sleep mode
can_doze_mode_exit	Exit sleep mode
can_error_type_record_get	Read CAN error type
can_receive_error_counter_get	Read CAN receive error counter
can_transmit_error_counter_get	Read CAN transmit error counter
can_interrupt_enable	Enable the selected CAN interrupt
can_flag_get	Read the selected CAN flag
Can_interrupt_flag_get	Read the selected CAN interrupt flag
can_flag_clear	Clear the selected CAN flag

### 5.3.1 can\_reset function

The table below describes the function can\_reset.

Table 57. can\_reset function

Name	Description
Function name	can_reset
Function prototype	void can_reset(can_type* can_x);
Function description	Reset CAN registers to their default values
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

**Example:**

```
can_reset(CAN1);
```

### 5.3.2 can\_baudrate\_default\_para\_init function

The table below describes the function can\_baudrate\_default\_para\_init.

Table 58. can\_baudrate\_default\_para\_init function

Name	Description
Function name	can_baudrate_default_para_init
Function prototype	void can_baudrate_default_para_init(can_baudrate_type* can_baudrate_struct);
Function description	Configure the CAN baud rate initial structure with the initial value
Input parameter 1	can_baudrate_struct: <a href="#">can_baudrate_type</a> pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of the can_baudrate_type before starting.
Called functions	NA

**Example:**

```
can_baudrate_type can_baudrate_struct;  
can_baudrate_default_para_init(&can_baudrate_struct);
```

### 5.3.3 can\_baudrate\_set function

The table below describes the function can\_baudrate\_set.

**Table 59. can\_baudrate\_set function**

Name	Description
Function name	can_baudrate_set
Function prototype	error_status can_baudrate_set(can_type* can_x, can_baudrate_type* can_baudrate_struct);
Function description	Set baud rate for CAN
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	can_baudrate_struct: <a href="#">can_baudrate_type</a> pointer
Output parameter	NA
Return value	status_index: check if baud rate is configured successfully
Required preconditions	It is necessary to define a variable of the can_baudrate_type before starting.
Called functions	NA

can\_baudrate\_type is defined in the at32f403\_can.h:

typedef struct

```
{
    uint16_t      baudrate_div;
    can_rsaw_type rsaw_size;
    can_bts1_type bts1_size;
    can_bts2_type bts2_size;
}
```

**can\_baudrate\_type;**

**baudrate\_div**

CAN clock division factor

Value range: 0x001~0x400

**rsaw\_size**

Defines the maximum of time unit that the CAN is allowed to lengthen or shorten in a bit

CAN\_RSAW\_1TQ: Resynchronization width is 1 time unit

CAN\_RSAW\_2TQ: Resynchronization width is 2 time units

CAN\_RSAW\_3TQ: Resynchronization width is 3 time units

CAN\_RSAW\_4TQ: Resynchronization width is 4 time units

**bts1\_size**

segment1 time duration

**bts1\_size** description

CAN\_BTS1\_1TQ: the bit time segment 1 has 1 time unit

...

CAN\_BTS1\_16TQ: the bit time segment 16 time units

**bts2\_size**

segment2 time duration

CAN\_BTS2\_1TQ: the bit time segment 2 has 1 time unit

...

CAN\_BTS2\_8TQ: the bit time segment 2 has 8 time units

**Example:**

```
/* can baudrate, set baudrate = pclk/(baudrate_div *(1 + bts1_size + bts2_size)) */
can_baudrate_struct.baudrate_div = 10;
can_baudrate_struct.rsaw_size = CAN_RSAW_3TQ;
can_baudrate_struct.bts1_size = CAN_BTS1_8TQ;
can_baudrate_struct.bts2_size = CAN_BTS2_3TQ;
can_baudrate_set(CAN1, &can_baudrate_struct);
```

### 5.3.4 can\_default\_para\_init function

The table below describes the function can\_default\_para\_init.

**Table 60. can\_default\_para\_init function**

Name	Description
Function name	can_default_para_init
Function prototype	void can_default_para_init(can_base_type* can_base_struct);
Function description	Set an initial value for CAN initial structure
Input parameter 1	can_base_struct: <i>can_base_type</i> pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of the can_base_type before starting.
Called functions	NA

**Example:**

```
can_base_type can_base_struct;
can_default_para_init (&can_base_struct);
```

### 5.3.5 can\_base\_init function

The table below describes the function can\_base\_init.

**Table 61. can\_base\_init function**

Name	Description
Function name	can_base_init
Function prototype	error_status can_base_init(can_type* can_x, can_base_type* can_base_struct);
Function description	Initialize CAN registers with the specified parameters in the can_base_struct
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	can_base_struct: <i>can_base_type</i> pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of can_base_type before starting.
Called functions	NA

The can\_base\_type is defined in the at32f403\_can.h:

```
typedef struct
{
    can_mode_type          mode_selection;
    confirm_state           ttc_enable;
    confirm_state           aebo_enable;
```

```
    confirm_state           aed_enable;
    confirm_state           prsf_enable;
    can_msg_discarding_rule_type mdrsel_selection;
    can_msg_sending_rule_type mmssr_selection;
} can_base_type;

mode_selection
Test mode selection
CAN_MODE_COMMUNICATE:          Communication mode
CAN_MODE_LOOPBACK:             Loopback mode
CAN_MODE_LISTENONLY:            Listen only mode
CAN_MODE_LISTENONLY_LOOPBACK:   Loopback + listen only mode

ttc_enable
Enable/disable time-triggered communication mode.
FALSE: Disable time-triggered communication mode
TRUE:  Enable time-triggered communication mode (while receiving/sending messages, capture time stamp and store it into the CAN RFCx and CAN TMCx registers)

aebo_enable
Enable auto exit of bus-off mode.
FALSE: Automatic exit of bus-off mode is disabled
TRUE:  Automatic exit of bus-off mode is enabled

aed_enable
Enable auto exit of sleep mode.
FALSE: Auto exit of sleep mode is disabled
TRUE:  Auto exit of sleep mode is enabled

prsf_enable
Disable retransmission when transmit failed.
FALSE: Retransmission is enabled.
TRUE:  Retransmission is disabled

mdrsel_selection
Define message discard rule when reception overflows.
CAN_DISCARDING_FIRST RECEIVED: The previous message is discarded.
CAN_DISCARDING_LAST RECEIVED: The new incoming message is discarded.

mmssr_selection
Define multiple message transmit sequence rule.
CAN_SENDING_BY_ID: The message with the smallest identifier is first transmitted.
CAN_SENDING_BY_REQUEST: The message with the first request order is first transmitted.

Example:
/* can base init */
can_base_struct.mode_selection = CAN_MODE_COMMUNICATE;
can_base_struct.ttc_enable = FALSE;
can_base_struct.aebo_enable = TRUE;
can_base_struct.aed_enable = TRUE;
can_base_struct.prf_enable = FALSE;
can_base_struct.mdrsel_selection = CAN_DISCARDING_FIRST RECEIVED;
can_base_struct.mmssr_selection = CAN_SENDING_BY_ID;
can_base_init(CAN1, &can_base_struct);
```

### 5.3.6 can\_filter\_default\_para\_init function

The table below describes the function can\_filter\_default\_para\_init.

**Table 62. can\_filter\_default\_para\_init function**

Name	Description
Function name	can_filter_default_para_init
Function prototype	void can_filter_default_para_init(can_filter_init_type* can_filter_init_struct);
Function description	Configure CAN filter initialization structure with the initial value
Input parameter 1	can_filter_init_struct: <a href="#">can_filter_init_type</a> pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of can_filter_init_type before starting.
Called functions	NA

**Example:**

```
can_filter_init_type can_filter_init_struct;
can_filter_default_para_init(&can_filter_init_struct);
```

### 5.3.7 can\_filter\_init function

The table below describes the function can\_filter\_init.

**Table 63. can\_filter\_init function**

Name	Description
Function name	can_filter_init
Function prototype	void can_filter_init(can_type* can_x, can_filter_init_type* can_filter_init_struct);
Function description	Initialize all CAN registers with the specified parameters in the can_base_struct
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1, CAN2
Input parameter 2	can_filter_init_struct: <a href="#">can_filter_init_type</a> pointer
Output parameter	NA
Return value	NA
Required preconditions	It is required to define a variable of the can_filter_init_type before starting.
Called functions	NA

can\_filter\_init\_type is defined in the at32f403\_can.h:

```
typedef struct
{
    confirm_state          filter_activate_enable;
    can_filter_mode_type   filter_mode;
    can_filter_fifo_type   filter_fifo;
    uint8_t                 filter_number;
    can_filter_bit_width_type filter_bit;
    uint16_t                filter_id_high;
    uint16_t                filter_id_low;
    uint16_t                filter_mask_high;
    uint16_t                filter_mask_low;
} can_filter_init_type;
```

**filter\_activate\_enable**

Enable/disable filter bank

FALSE: Disable filter bank

TRUE: Enable filter bank

**filter\_mode**

Select filter mode.

CAN\_FILTER\_MODE\_ID\_MASK: Identifier mask mode

CAN\_FILTER\_MODE\_ID\_LIST: Identifier list mode

**filter\_fifo**

Select filter associated FIFO.

CAN\_FILTER\_FIFO0: Associated with FIFO0

CAN\_FILTER\_FIFO1: Associated with FIFO1

**filter\_number**

Select filter bank.

Value range: 0~13

**filter\_bit**

Select filter bit width

CAN\_FILTER\_16BIT: 16-bit

CAN\_FILTER\_32BIT: 32-bit

**filter\_id\_high**

The filter\_id\_high is used to configure the upper 16 bits (32-bit width, Mask/List mode) of the filter identifier 1, the filter identifier 2 (16-bit width, List mode) or the filter mask identifier 1 (16-bit width, Mask mode)

Value range: 0x0000~0xFFFF

**filter\_id\_low**

The filter\_id\_low is used to configure the lower 16 bits of the filter identifier 1 (32-bit width, Mask/List mode), or the filter identifier 1 (16-bit width, List mode).

Value range: 0x0000~0xFFFF

**filter\_mask\_high**

The filter\_mask\_high is used to configure the upper 16 bits of the filter mask identifier 1 (32-bit width, Mask mode), the filter mask identifier 2 (16-bit width, Mask mode), the upper 16 bits of the filter identifier 2 (32-bit width, List mode) or the filter identifier 4 (16-bit width, List mode)

Value range: 0x0000~0xFFFF

**filter\_mask\_low**

The filter\_mask\_low is used to configure the lower 16 bits of the filter mask identifier 1 (32-bit width, Mask mode), the filter identifier 2 (16-bit width, Mask mode), the lower 16 bits of the filter identifier 2 (32-bit width ,List mode) or the filter identifier 3 (16-bit width, List mode)

Value range: 0x0000~0xFFFF

**Example:**

```
/* can filter init */  
can_filter_init_struct.filter_activate_enable = TRUE;  
can_filter_init_struct.filter_mode = CAN_FILTER_MODE_ID_MASK;  
can_filter_init_struct.filter_fifo = CAN_FILTER_FIFO0;  
can_filter_init_struct.filter_number = 0;  
can_filter_init_struct.filter_bit = CAN_FILTER_32BIT;  
can_filter_init_struct.filter_id_high = 0;
```

```

can_filter_init_struct.filter_id_low = 0;
can_filter_init_struct.filter_mask_high = 0;
can_filter_init_struct.filter_mask_low = 0;
can_filter_init(CAN1, &can_filter_init_struct);

```

### 5.3.8 can\_debug\_transmission\_prohibit function

The table below describes the function can\_debug\_transmission\_prohibit.

**Table 64. can\_debug\_transmission\_prohibit function**

Name	Description
Function name	can_debug_transmission_prohibit
Function prototype	void can_debug_transmission_prohibit(can_type* can_x, confirm_state new_state);
Function description	Disable/enable message transceiver when debugging
Input parameter 1	adc_x: indicates the selected CAN This parameter can be any CAN1
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```

/* prohibit can trans when debug*/
can_debug_transmission_prohibit(CAN1, TRUE);

```

### 5.3.9 can\_ttc\_mode\_enable function

The table below describes the function can\_ttc\_mode\_enable.

**Table 65. can\_ttc\_mode\_enable function**

Name	Description
Function name	can_ttc_mode_enable
Function prototype	void can_ttc_mode_enable(can_type* can_x, confirm_state new_state);
Function description	Enable time-triggered mode
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```

/* can time trigger operation communication mode enable*/
can_ttc_mode_enable (CAN1, TRUE);

```

*Note: When the ttc\_enable is enabled in the can\_base\_init, it indicates that only the time stamp is enabled (During message receive and transmit, the time stamp is captured and stored in the CAN\_RFCx and CAN\_TMCx registers). But when the can\_ttc\_mode\_enable is enabled, not only the time stamp is enabled, and but the time stamp transmission feature is enabled (During message transmission, the time stamp is sent on the 7<sup>th</sup> and 8<sup>th</sup> data byte).*

### 5.3.10 can\_message\_transmit function

The table below describes the function can\_message\_transmit.

**Table 66. can\_message\_transmit function**

Name	Description
Function name	can_message_transmit
Function prototype	uint8_t can_message_transmit(can_type* can_x, can_tx_message_type* tx_message_struct);
Function description	Transmit a frame of message
Input parameter 1	adc_x: indicates the selected CAN This parameter can be any CAN1
Input parameter 2	tx_message_struct: message pending for transmission, refer to <a href="#">can_tx_message_type</a>
Output parameter	NA
Return value	transmit_mailbox: indicates the mailbox number required to send message
Required preconditions	Write the to-be-sent message in the tx_message_struct
Called functions	NA

can\_tx\_message\_type is defined in the at32f403\_can.h:

typedef struct

```
{
    uint32_t          standard_id;
    uint32_t          extended_id;
    can_identifier_type id_type;
    can_trans_frame_type frame_type;
    uint8_t            dlc;
    uint8_t            data[8];
}
```

} can\_tx\_message\_type;

**standard\_id**

Standard identifier (11 bits active)

Value range: 0x000~0x7FF

**extended\_id**

Extended identifier (29 bits active)

Value range: 0x000~0x1FFFFFF

**id\_type**

Identifier type

CAN\_ID\_STANDARD: Standard identifier

CAN\_ID\_EXTENDED: Extended identifier

**frame\_type**

Frame type

CAN\_TFT\_DATA: Data frame

CAN\_TFT\_REMOTE: Remote frame

**dlc**

Data length (in byte)

Value range: 0~8

**data[8]**

Data pending for transmission

Value range: 0x00~0xFF

**Example:**

```

/* can transmit data */

static void can_transmit_data(void)
{
    uint8_t transmit_mailbox;
    can_tx_message_type tx_message_struct;
    tx_message_struct.standard_id = 0x400;
    tx_message_struct.extended_id = 0;
    tx_message_struct.id_type = CAN_ID_STANDARD;
    tx_message_struct.frame_type = CAN_TFT_DATA;
    tx_message_struct.dlc = 8;
    tx_message_struct.data[0] = 0x11;
    tx_message_struct.data[1] = 0x22;
    tx_message_struct.data[2] = 0x33;
    tx_message_struct.data[3] = 0x44;
    tx_message_struct.data[4] = 0x55;
    tx_message_struct.data[5] = 0x66;
    tx_message_struct.data[6] = 0x77;
    tx_message_struct.data[7] = 0x88;
    transmit_mailbox = can_message_transmit(CAN1, &tx_message_struct);
    while(can_transmit_status_get(CAN1, (can_tx_mailbox_num_type)transmit_mailbox) != CAN_TX_STATUS_SUCCESSFUL);
}

```

### 5.3.11 can\_transmit\_status\_get function

The table below describes the function can\_transmit\_status\_get.

**Table 67. can\_transmit\_status\_get function**

Name	Description
Function name	can_transmit_status_get
Function prototype	can_transmit_status_type can_transmit_status_get(can_type* can_x, can_tx_mailbox_num_type transmit_mailbox);
Function description	Get the status of transmission
Input parameter 1	adc_x: indicates the selected CAN This parameter can be any CAN1
Input parameter 2	transmit_mailbox: indicates the mailbox number required to send message
Output parameter	NA
Return value	state_index: transmission status
Required preconditions	First send a frame of message and get a transmit mailbox number
Called functions	NA

**Example:**

```

/* can transmit data */

static void can_transmit_data(void)
{
    uint8_t transmit_mailbox;
    can_tx_message_type tx_message_struct;

```

```

tx_message_struct.standard_id = 0x400;
tx_message_struct.extended_id = 0;
tx_message_struct.id_type = CAN_ID_STANDARD;
tx_message_struct.frame_type = CAN_TFT_DATA;
tx_message_struct.dlc = 8;
tx_message_struct.data[0] = 0x11;
tx_message_struct.data[1] = 0x22;
tx_message_struct.data[2] = 0x33;
tx_message_struct.data[3] = 0x44;
tx_message_struct.data[4] = 0x55;
tx_message_struct.data[5] = 0x66;
tx_message_struct.data[6] = 0x77;
tx_message_struct.data[7] = 0x88;
transmit_mailbox = can_message_transmit(CAN1, &tx_message_struct);
while(can_transmit_status_get(CAN1, (can_tx_mailbox_num_type)transmit_mailbox) != CAN_TX_STATUS_SUCCESSFUL);
}

```

### 5.3.12 can\_transmit\_cancel function

The table below describes the function can\_transmit\_cancel.

**Table 68. can\_transmit\_cancel function**

Name	Description
Function name	can_transmit_cancel
Function prototype	void can_transmit_cancel(can_type* can_x, can_tx_mailbox_num_type transmit_mailbox);
Function description	Cancel transmission
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	transmit_mailbox: indicates the mailbox number required to send message
Output parameter	NA
Return value	NA
Required preconditions	First send a frame of message and get a transmit mailbox number
Called functions	NA

**Example:**

```

/* cancel a transmit request */
uint8_t transmit_mailbox;
transmit_mailbox = can_message_transmit(CAN1, &tx_message_struct);
can_transmit_cancel(CAN1, (can_tx_mailbox_num_type)transmit_mailbox);

```

### 5.3.13 can\_message\_receive function

The table below describes the function can\_message\_receive.

**Table 69. can\_message\_receive function**

Name	Description
Function name	can_message_receive
Function prototype	void can_message_receive(can_type* can_x, can_rx_fifo_num_type fifo_number, can_rx_message_type* rx_message_struct);
Function description	Receive message
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	fifo_number: receive FIFO This parameter can be CAN_RX_FIFO0 or CAN_RX_FIFO1.
Output parameter	rx_message_struct: indicates the received message, refer to <a href="#">can_rx_message_type</a>
Return value	NA
Required preconditions	Receive FIFO not empty (FIFO message count is not zero)
Called functions	void can_receive_fifo_release(can_type* can_x, can_rx_fifo_num_type fifo_number);

can\_rx\_message\_type is defined in the at32f403\_can.h:

typedef struct

```
{
    uint32_t          standard_id;
    uint32_t          extended_id;
    can_identifier_type id_type;
    can_trans_frame_type frame_type;
    uint8_t            dlc;
    uint8_t            data[8];
    uint8_t            filter_index;
}
```

} can\_rx\_message\_type;

**standard\_id**

Standard identifier (11 bit active)

Value range: 0x000~0x7FF

**extended\_id**

Extended identifier (29 bits active)

Value range: 0x000~0x1FFFFFFF

**id\_type**

Identifier type

CAN\_ID\_STANDARD: Standard identifier

CAN\_ID\_EXTENDED: Extended identifier

**frame\_type**

Frame type

CAN\_TFT\_DATA: Data frame

CAN\_TFT\_REMOTE: Remote frame

**dlc**

Data length (in byte)

Value range: 0~8

**data[8]**

Data pending for transmission

Value range: 0x00~0xFF

#### **filter\_index**

Filter match index (indicating the filter number that a message has passed through.)

Value range: 0x00~0xFF

#### **Example:**

```
/* can receive message */
can_rx_message_type rx_message_struct;
can_message_receive(CAN1, CAN_RX_FIFO0, &rx_message_struct);
```

### **5.3.14 can\_receive\_fifo\_release function**

The table below describes the function can\_receive\_fifo\_release.

**Table 70. can\_receive\_fifo\_release function**

Name	Description
Function name	can_receive_fifo_release
Function prototype	void can_receive_fifo_release(can_type* can_x, can_rx_fifo_num_type fifo_number);
Function description	Release receive FIFO
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	fifo_number: receive FIFO number This parameter can be either CAN_RX_FIFO0 or CAN_RX_FIFO1
Output parameter	NA
Return value	NA
Required preconditions	Message in FIFO has already been read
Called functions	NA

#### **Example:**

```
/* can receive message */
void can_message_receive(can_type* can_x, can_rx_fifo_num_type fifo_number, can_rx_message_type*
rx_message_struct)
{
    /* get the id type */
    rx_message_struct->id_type = (can_identifier_type)can_x->fifo_mailbox[fifo_number].rfi_bit.rfidi;
    ...

    /* get the data field */
    rx_message_struct->data[0] = can_x->fifo_mailbox[fifo_number].rfdtl_bit.rfdt0;
    ...
    rx_message_struct->data[7] = can_x->fifo_mailbox[fifo_number].rfdth_bit.rfdt7;

    /*FIFO must be read before releasing FIFO */
    /* release the fifo */
    can_receive_fifo_release(can_x, fifo_number);
}
```

### 5.3.15 can\_receive\_message\_pending\_get function

The table below describes the function can\_receive\_message\_pending\_get.

**Table 71. can\_receive\_message\_pending\_get function**

Name	Description
Function name	can_receive_message_pending_get
Function prototype	uint8_t can_receive_message_pending_get(can_type* can_x, can_rx_fifo_num_type fifo_number);
Function description	Get the number of message pending for read in FIFO
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	fifo_number: receive FIFO number This parameter can be either CAN_RX_FIFO0 or CAN_RX_FIFO1
Output parameter	NA
Return value	message_pending: the count of message pending for read in FIFO
Required preconditions	NA
Called functions	NA

**Example:**

```
/* return the number of pending messages of */
can_receive_message_pending_get (CAN1, CAN_RX_FIFO0);
```

### 5.3.16 can\_operating\_mode\_set function

The table below describes the function can\_operating\_mode\_set.

**Table 72. can\_operating\_mode\_set function**

Name	Description
Function name	can_operating_mode_set
Function prototype	error_status can_operating_mode_set(can_type* can_x, can_operating_mode_type can_operating_mode);
Function description	Configure CAN operating modes
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	<a href="#">can_operating_mode</a> : CAN operating mode selection
Output parameter	NA
Return value	Status: indicates whether configuration is successful or not
Required preconditions	NA
Called functions	NA

**can\_operating\_mode**

CAN\_OPERATINGMODE\_FREEZE: Freeze mode—for CAN controller initialization

CAN\_OPERATINGMODE\_DOZE: Sleep mode—CAN clock stopped to save power consumption

CAN\_OPERATINGMODE\_COMMUNICATE: Communication mode—for communication

**Example:**

```
/* set the operation mode —enter freeze mode*/
can_operating_mode_set (CAN1, CAN_OPERATINGMODE_FREEZE);
```

```
/*Initialize CAN controller*/  
...  
  
/* set the operation mode -enter communicate mode*/  
can_operating_mode_set (CAN1, CAN_OPERATINGMODE_COMMUNICATE);  
  
/*Starts communication: send and receive message */  
...
```

### 5.3.17 can\_doze\_mode\_enter function

The table below describes the function can\_doze\_mode\_enter

Table 73. can\_doze\_mode\_enter function

Name	Description
Function name	can_doze_mode_enter
Function prototype	can_enter_doze_status_type can_doze_mode_enter(can_type* can_x);
Function description	Enter sleep mode
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Output parameter	NA
Return value	<a href="#">can_enter_doze_status</a> : indicates whether the Sleep mode is entered
Required preconditions	NA
Called functions	NA

#### can\_enter\_doze\_status

Indicates whether the Sleep mode is entered or not

CAN\_ENTER\_DOZE\_FAILED: Sleep mode entry failure

CAN\_ENTER\_DOZE\_SUCCESSFUL: Sleep mode entry success

#### Example:

```
/* can enter the low power mode */  
can_enter_doze_status_type can_enter_doze_status;  
can_enter_doze_status = can_doze_mode_enter(CAN1);
```

### 5.3.18 can\_doze\_mode\_exit function

The table below describes the function can\_doze\_mode\_exit.

**Table 74. can\_doze\_mode\_exit function**

Name	Description
Function name	can_doze_mode_exit
Function prototype	can_quit_doze_status_type can_doze_mode_exit(can_type* can_x);
Function description	Exit Sleep mode
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Output parameter	NA
Return value	<i>can_quit_doze_status</i> : indicates whether the Sleep mode has been left
Required preconditions	NA
Called functions	NA

#### can\_quit\_doze\_status

Indicates whether the Sleep mode has been left successfully.

CAN\_QUIT\_DOZE\_FAILED: Sleep mode exit failure

CAN\_QUIT\_DOZE\_SUCCESSFUL: Sleep mode exit success

#### Example:

```
/* can exit the low power mode */
can_quit_doze_status_type can_quit_doze_status;
can_quit_doze_status = can_doze_mode_exit (CAN1);
```

### 5.3.19 can\_error\_type\_record\_get function

The table below describes the function can\_error\_type\_record\_get.

**Table 75. can\_error\_type\_record\_get function**

Name	Description
Function name	can_error_type_record_get
Function prototype	can_error_record_type can_error_type_record_get(can_type* can_x);
Function description	Read CAN error type
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Output parameter	NA
Return value	<i>can_error_record</i> : Error type
Required preconditions	NA
Called functions	NA

#### can\_error\_record

CAN error record

CAN_ERRORRECORD_NOERR:	No error
CAN_ERRORRECORD_STUFFERR:	Bit stuffing error
CAN_ERRORRECORD_FORMERR:	Format error
CAN_ERRORRECORD_ACKERR:	Acknowledge error
CAN_ERRORRECORD_BITRECESSIVEERR:	Recessive bit error
CAN_ERRORRECORD_BITDOMINANTERR:	Dominant bit error
CAN_ERRORRECORD_CRCERR:	CRC error

CAN\_ERRORRECORD\_SOFTWARESETERR: Set by software

**Example:**

```
/* get the error type record (etr) */
can_error_record_type can_error_record;
can_error_record = can_error_type_record_get (CAN1);
```

### 5.3.20 can\_receive\_error\_counter\_get function

The table below describes the function can\_receive\_error\_counter\_get.

**Table 76. can\_receive\_error\_counter\_get function**

Name	Description
Function name	can_receive_error_counter_get
Function prototype	uint8_t can_receive_error_counter_get(can_type* can_x);
Function description	Read CAN receive error counter
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Output parameter	NA
Return value	receive_error_counter: Receive error counter Value range: 0x00~0xFF
Required preconditions	NA
Called functions	NA

**Example:**

```
/* get the receive error counter (rec) */
uint8_t receive_error_counter;
receive_error_counter = can_receive_error_counter_get (CAN1);
```

### 5.3.21 can\_transmit\_error\_counter\_get function

The table below describes the function can\_transmit\_error\_counter\_get.

**Table 77. can\_transmit\_error\_counter\_get function**

Name	Description
Function name	can_transmit_error_counter_get
Function prototype	uint8_t can_transmit_error_counter_get(can_type* can_x);
Function description	Read CAN transmit error counter
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Output parameter	NA
Return value	transmit_error_counter: Transmit error counter Value range:0x00~0xFF
Required preconditions	NA
Called functions	NA

**Example:**

```
/* get the transmit error counter (tec) */
uint8_t transmit_error_counter;
transmit_error_counter = can_transmit_error_counter_get (CAN1);
```

### 5.3.22 can\_interrupt\_enable function

The table below describes the function can\_interrupt\_enable.

**Table 78. can\_interrupt\_enable function**

Name	Description
Function name	can_interrupt_enable
Function prototype	void can_interrupt_enable(can_type* can_x, uint32_t can_int, confirm_state new_state);
Function description	Enable the selected CAN interrupt
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	<i>can_int</i> : Select CAN interrupts
Input parameter 3	new_state: Enable or disable This parameter can be FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### can\_int

CAN interrupt select

CAN_TCIEN_INT:	Transmit mailbox empty interrupt enable
CAN_RF0MIEN_INT:	FIFO 0 receive message interrupt enable
CAN_RF0FIEN_INT:	Receive FIFO0 full interrupt enable
CAN_RF0OIEN_INT:	Receive FIFO0 overflow interrupt enable
CAN_RF1MIEN_INT:	FIFO 1 receive message interrupt enable
CAN_RF1FIEN_INT:	Receive FIFO1 full interrupt enable
CAN_RF1OIEN_INT:	Receive FIFO1 overflow interrupt enable
CAN_EAIEN_INT:	Error active interrupt enable
CAN_EPIEN_INT:	Error passive interrupt enable
CAN_BOIEN_INT:	Bus-off interrupt enable
CAN_ETRIEN_INT:	Error type record interrupt enable
CAN_EOIEN_INT:	Error occur interrupt enable
CAN_QDZIEN_INT:	Quit Sleep mode interrupt enable
CAN_EDZIEN_INT:	Enter Sleep mode interrupt enable

#### Example

```

/* can interrupt config */
nvic_irq_enable(CAN1_SE_IRQn, 0x00, 0x00);/*CAN1 error/status change interrupt*/
nvic_irq_enable(USBFS_L_CAN1_RX0_IRQn, 0x00, 0x00);/*CAN1 FIFO0 receive interrupt*/

/* FIFO 0 receive message interrupt enable */
can_interrupt_enable(CAN1, CAN_RF0MIEN_INT, TRUE);

/* error type record interrupt enable */
can_interrupt_enable(CAN1, CAN_ETRIEN_INT, TRUE);

/*This parameter is an error interrupt controller and it is enabled before error-related interrupts */
can_interrupt_enable(CAN1, CAN_EOIEN_INT, TRUE);

```

### 5.3.23 can\_flag\_get function

The table below describes the function can\_flag\_get.

**Table 79. can\_flag\_get function**

Name	Description
Function name	can_flag_get
Function prototype	flag_status can_flag_get(can_type* can_x, uint32_t can_flag);
Function description	Get the status of the selected CAN flag
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	<i>can_flag</i> : indicates the selected flag Refer to the “can_flag” description below for details.
Output parameter	NA
Return value	flag_status: the status of the selected flag Return value can be RESET or SET.
Required preconditions	NA
Called functions	NA

#### can\_flag

This is used to select a flag and get its status, including:

- CAN\_EAF\_FLAG: Error active flag
- CAN\_EPFLAG: Error passive flag
- CAN\_BOFFLAG: Bus-off flag
- CAN\_ETR\_FLAG: Error type record (non-zero Error type flag)
- CAN\_EOIF\_FLAG: Error occur interrupt flag
- CAN\_TMF0TCF\_FLAG: Mailbox 0 transmission complete flag
- CAN\_TMF1TCF\_FLAG: Mailbox 1 transmission complete flag
- CAN\_TMF2TCF\_FLAG: Mailbox 2 transmission complete flag
- CAN\_RF0MN\_FLAG: Receive FIFO0 non-empty flag
- CAN\_RF0FF\_FLAG: FIFO0 full flag
- CAN\_RF0OF\_FLAG: FIFO0 overflow flag
- CAN\_RF1MN\_FLAG: FIFO1 non-empty flag
- CAN\_RF1FF\_FLAG: FIFO1 full flag
- CAN\_RF1OF\_FLAG: FIFO1 overflow flag
- CAN\_QDZIF\_FLAG: Quite Sleep mode flag
- CAN\_EDZC\_FLAG: Enter Sleep mode flag
- CAN\_TMEF\_FLAG: Transmit mailbox empty flag (any one of three transmit mailboxes is empty)

#### Example:

```
/* get receive fifo 0 message num flag */
flag_status bit_status = RESET;
bit_status = can_flag_get (CAN1, CAN_RF0MN_FLAG);
```

### 5.3.24 can\_interrupt\_flag\_get function

The table below describes the function can\_interrupt\_flag\_get.

**Table 80. can\_interrupt\_flag\_get function**

Name	Description
Function name	can_interrupt_flag_get
Function prototype	flag_status can_interrupt_flag_get(can_type* can_x, uint32_t can_flag);
Function description	Get CAN interrupt flag
Input parameter 1	can_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	<a href="#">can_flag</a> : indicates the selected flag Refer to the “can_flag” description below for details.
Output parameter	NA
Return value	flag_status: the status of the selected flag Return value can be RESET or SET
Required preconditions	NA
Called functions	NA

#### can\_flag

This is used to select a flag and get its status, including:

- CAN\_EAF\_FLAG: Error active flag
- CAN\_EPFLAG: Error passive flag
- CAN\_BOFFLAG: Bus-off flag
- CAN\_ETR\_FLAG: Error type record (non-zero Error type flag)
- CAN\_EOIF\_FLAG: Error occur interrupt flag
- CAN\_TMF0TCF\_FLAG: Mailbox 0 transmission complete flag
- CAN\_TMF1TCF\_FLAG: Mailbox 1 transmission complete flag
- CAN\_TMF2TCF\_FLAG: Mailbox 2 transmission complete flag
- CAN\_RF0MN\_FLAG: FIFO0 non-empty flag
- CAN\_RF0FF\_FLAG: FIFO0 full flag
- CAN\_RF0OF\_FLAG: FIFO0 overflow flag
- CAN\_RF1MN\_FLAG: FIFO1 non-empty flag
- CAN\_RF1FF\_FLAG: FIFO1 full flag
- CAN\_RF1OF\_FLAG: FIFO1 overflow flag
- CAN\_QDZIF\_FLAG: Quite Sleep mode flag
- CAN\_EDZC\_FLAG: Enter Sleep mode flag
- CAN\_TMEF\_FLAG: Transmit mailbox empty flag (any one of three transmit mailboxes is empty)

#### Example

```
/* check receive fifo 0 message num interrupt flag */
if(can_interrupt_flag_get(CAN1, CAN_RF0MN_FLAG) != RESET)
{
}
```

### 5.3.25 can\_flag\_clear function

The table below describes the function can\_flag\_clear.

**Table 81. can\_flag\_clear function**

Name	Description
Function name	can_flag_clear
Function prototype	void can_flag_clear(can_type* can_x, uint32_t can_flag);
Function description	Clear the selected CAN flag
Input parameter 1	adc_x: indicates the selected CAN This parameter can be CAN1
Input parameter 2	can_flag: indicates the selected flag Refer to the "can_flag" description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### can\_flag:

This is used to clear the selected flag, including:

- CAN\_EAF\_FLAG: Error active flag
- CAN\_EPF\_FLAG: Error passive flag
- CAN\_BOF\_FLAG: Bus-off flag
- CAN\_ETR\_FLAG: Error type record (non-zero Error type flag)
- CAN\_EOIF\_FLAG: Error occur interrupt flag
- CAN\_TM0TCF\_FLAG: Mailbox 0 transmission complete flag
- CAN\_TM1TCF\_FLAG: Mailbox 1 transmission complete flag
- CAN\_TM2TCF\_FLAG: Mailbox 2 transmission complete flag
- CAN\_RF0FF\_FLAG: FIFO0 full flag
- CAN\_RF0OF\_FLAG: FIFO0 overflow flag
- CAN\_RF1FF\_FLAG: FIFO1 full flag
- CAN\_RF1OF\_FLAG: FIFO1 overflow flag
- CAN\_QDZIF\_FLAG: Leave Sleep mode flag
- CAN\_EDZC\_FLAG: Enter Sleep mode flag
- CAN\_TMEF\_FLAG: Transmit mailbox empty flag (any one of three transmit mailboxes is empty)

*Note: The CAN\_RF0MN\_FLAG (FIFO0 non-empty flag) and CAN\_RF1MN\_FLAG (FIFO1 non-empty flag) have no clear operations since both are defined by software.*

#### Example:

```
/* clear receive fifo 0 overflow flag */
can_flag_clear (CAN1, CAN_RF1OF_FLAG);
```

## 5.4 CRC calculation unit (CRC)

The CRC register structure `crc_type` is defined in the “`at32f403_crc.h`”:

```
/*
 * @brief type define crc register all
 */
typedef struct
{
    ...
} crc_type;
```

The table below gives a list of the CRC registers.

**Table 82. Summary of CRC registers**

Register	Description
dt	Data register
cdt	General-purpose data register
ctrl	Control register
idt	Initialization register
poly	Polynomial generation register

The table below gives a list of CRC library functions.

**Table 83. Summary of CRC library functions**

Function name	Description
<code>crc_data_reset</code>	Data register reset
<code>crc_one_word_calculate</code>	Calulate the CRC value using combination of a new 32-bit data and the previous CRC value
<code>crc_block_calculate</code>	Write a data block in order into CRC check and return the calculated result
<code>crc_data_get</code>	Get the currently calculated CRC result
<code>crc_common_data_set</code>	Configure common registers
<code>crc_common_data_get</code>	Get the value of common registers
<code>crc_init_data_set</code>	Set the CRC initialization register
<code>crc_reverse_input_data_set</code>	Set CRC input data bit reverse type
<code>crc_reverse_output_data_set</code>	Set CRC output data reverse type
<code>crc_poly_value_set</code>	Set polynomial value
<code>crc_poly_value_get</code>	Get polynomial value
<code>crc_poly_size_set</code>	Set polynomial size
<code>crc_poly_size_get</code>	Get polynomial size

## 5.4.1 crc\_data\_reset function

The table below describes the function crc\_data\_reset.

**Table 84. crc\_data\_reset function**

Name	Description
Function name	crc_data_reset
Function prototype	void crc_data_reset(void);
Function description	When the data register is reset, the value of the initialization register is added into the data register as an initial value. The default reset value is 0xFFFFFFFF.
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* reset crc data register */  
crc_data_reset();
```

## 5.4.2 crc\_one\_word\_calculate function

The table below describes the function crc\_one\_word\_calculate.

**Table 85. crc\_one\_word\_calculate function**

Name	Description
Function name	crc_one_word_calculate
Function prototype	uint32_t crc_one_word_calculate(uint32_t data);
Function description	Calculate the CRC value using combination of a new 32-bit data and the previous CRC value
Input parameter 1	Data: input a 32-bit data
Input parameter 2	NA
Output parameter	NA
Return value	uint32_t: return CRC calculation result
Required preconditions	NA
Called functions	NA

**Example:**

```
/* calculate and return result */  
uint32_t data = 0x12345678, result = 0;  
result = crc_one_word_calculate (data);
```

### 5.4.3 crc\_block\_calculate function

The table below describes the function crc\_block\_calculate.

**Table 86. crc\_block\_calculate function**

Name	Description
Function name	crc_block_calculate
Function prototype	uint32_t crc_block_calculate(uint32_t *pbuffer, uint32_t length);
Function description	Input a data block in sequence to go through CRC calculation and return a result
Input parameter 1	Pbuffer: point to the data block pending for CRC check
Input parameter 2	Length: data block length pending for CRC check, in terms of 32-bit
Output parameter	NA
Return value	uint32_t: return CRC calculation result
Required preconditions	NA
Called functions	NA

**Example:**

```
/* calculate and return result */
uint32_t pbuffer[2] = {0x12345678, 0x87654321};
uint32_t result = 0;
result = crc_block_calculate (pbuffer, 2);
```

### 5.4.4 crc\_data\_get function

The table below describes the function crc\_data\_get.

**Table 87. crc\_data\_get function**

Name	Description
Function name	crc_data_get
Function prototype	uint32_t crc_data_get(void);
Function description	Return the current CRC calculation result
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	uint32_t: return CRC calculation result
Required preconditions	NA
Called functions	NA

**Example:**

```
/* get result */
uint32_t result = 0;
result = crc_data_get();
```

## 5.4.5 crc\_common\_data\_set function

The table below describes the function crc\_common\_data\_set.

**Table 88. crc\_common\_data\_set function**

Name	Description
Function name	crc_common_data_set
Function prototype	void crc_common_data_set(uint8_t cdt_value);
Function description	Configure common data register
Input parameter 1	cdt_value: 8-bit common data that can be used as temporary storage data
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* set common data */  
crc_common_data_set (0x88);
```

## 5.4.6 crc\_common\_date\_get function

The table below describes the function crc\_common\_data\_get.

**Table 89. crc\_common\_date\_get function**

Name	Description
Function name	crc_common_data_get
Function prototype	uint8_t crc_common_data_get(void);
Function description	Return the value of the command data register
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	uint8_t: return the value of the previously programmed common data register
Required preconditions	NA
Called functions	NA

**Example:**

```
/* get common data */  
uint8_t cdt_value = 0;  
cdt_value = crc_common_data_set ();
```

## 5.4.7 crc\_init\_data\_set function

The table below describes the function `crc_init_data_set`.

**Table 90. `crc_init_data_set` function**

Name	Description
Function name	<code>crc_init_data_set</code>
Function prototype	<code>void crc_init_data_set(uint32_t value);</code>
Function description	Set the value of the CRC initialization register
Input parameter 1	Value: the value of the CRC initialization register
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

After the value of the CRC initialization register is programmed, the CRC data register is updated with this value whenever the `crc_data_reset` function is called.

**Example:**

```
/* set initial data */
uint32_t init_value = 0x11223344;
crc_init_data_set (init_value);
```

## 5.4.8 crc\_reverse\_input\_data\_set function

The table below describes the function `crc_reverse_input_data_set`.

**Table 91. `crc_reverse_input_data_set` function**

Name	Description
Function name	<code>crc_reverse_input_data_set</code>
Function prototype	<code>void crc_reverse_input_data_set(crc_reverse_input_type value);</code>
Function description	Define the CRC input data bit reverse type
Input parameter 1	Value: input data bit reverse type
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**value**

Defines the reverse type of input data bit.

`CRC_REVERSE_INPUT_NO_AFFECTE`: No effect

`CRC_REVERSE_INPUT_BY_BYTE`: Byte reverse

`CRC_REVERSE_INPUT_BY_HALFWORD`: Half-word reverse

`CRC_REVERSE_INPUT_BY_WORD`: Word reverse

**Example:**

```
/* set input data reversing type */
crc_reverse_input_data_set(CRC_REVERSE_INPUT_BY_WORD);
```

## 5.4.9 crc\_reverse\_output\_data\_set function

The table below describes the function `crc_reverse_output_data_set`.

**Table 92. `crc_reverse_output_data_set` function**

Name	Description
Function name	<code>crc_reverse_output_data_set</code>
Function prototype	<code>void crc_reverse_output_data_set(crc_reverse_output_type value);</code>
Function description	Define the CRC output data reverse type
Input parameter 1	Value: output data bit reverse type
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### value

Defines the reverse type of output data bit.

CRC\_REVERSE\_OUTPUT\_NO\_AFFECTE: No effect

CRC\_REVERSE\_OUTPUT\_DATA: Word reverse

### Example:

```
/* set output data reversing type */
crc_reverse_output_data_set(CRC_REVERSE_OUTPUT_DATA);
```

## 5.4.10 crc\_poly\_value\_set function

The table below describes the function `crc_poly_value_set`.

**Table 93. `crc_poly_value_set` function**

Name	Description
Function name	<code>crc_poly_value_set</code>
Function prototype	<code>void crc_poly_value_set(uint32_t value);</code>
Function description	Set CRC polynomial value
Input parameter 1	value: polynomial value
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### Example

```
/* set poly value */
crc_poly_value_set(0x12345671);
```

## 5.4.11 crc\_poly\_value\_get function

The table below describes the function crc\_poly\_value\_get.

**Table 94. crc\_poly\_value\_get function**

Name	Description
Function name	crc_poly_value_get
Function prototype	uint32_t crc_poly_value_get(void);
Function description	Get CRC polynomial value
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	uint32_t: current polynomial value
Required preconditions	NA
Called functions	NA

### Example

```
/* get poly value */
uint32_t poly = 0;
poly = crc_poly_value_get();
```

## 5.4.12 crc\_poly\_size\_set function

The table below describes the function crc\_poly\_size\_set.

**Table 95. crc\_poly\_size\_set function**

Name	Description
Function name	crc_poly_size_set
Function prototype	void crc_poly_size_set(crc_poly_size_type size);
Function description	Set CRC polynomial size
Input parameter 1	size: valid polynomial width
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### size

This bit is used to define valid polynomial width.

CRC\_POLY\_SIZE\_32B: 32-bit valid width

CRC\_POLY\_SIZE\_16B: 16-bit valid width

CRC\_POLY\_SIZE\_8B: 8-bit valid width

CRC\_POLY\_SIZE\_7B: 7-bit valid width

### Example

```
/* set poly size 32-bit */
crc_poly_size_set(CRC_POLY_SIZE_32B);
```

## 5.4.13 crc\_poly\_size\_get function

The table below describes the function crc\_poly\_size\_get.

**Table 96. crc\_poly\_size\_get function**

Name	Description
Function name	crc_poly_size_get
Function prototype	crc_poly_size_type crc_poly_size_get(void);
Function description	Get CRC polynomial size
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	crc_poly_size_type: valid polynomial width
Required preconditions	NA
Called functions	NA

### crc\_poly\_size\_type

This bit is used to define valid polynomial width.

- CRC\_SIZE\_32B: 32-bit valid width
- CRC\_SIZE\_16B: 16-bit valid width
- CRC\_SIZE\_8B: 8-bit valid width
- CRC\_SIZE\_7B: 7-bit valid width

### Example

```
/* get poly size */  
crc_poly_size_type size;  
size = crc_poly_size_get();
```

## 5.5 Clock and reset management (CRM)

The CRM register structure `crm_type` is defined in the “`at32f403_crm.h`”:

```
/*
 * @brief type define crm register all
 */
typedef struct
{
    ...
} crm_type;
```

The table below gives a list of the CRM registers.

**Table 97. Summary of CRM registers**

Register	Description
ctrl	Clock control register
cfg	Clock configuration register
clkint	Clock interrupt register
apb2rst	APB2 peripheral reset register
apb1rst	APB1 peripheral reset register
ahben	AHB peripheral clock enable register
apb2en	APB2 peripheral clock register
apb1en	APB1 peripheral clock register
bpdc	Battery powered domain control register
ctrlsts	Control/status register
ahbrst	AHB peripheral reset register
misc1	Extra register 1
misc2	Extra register 2
misc3	Extra register 3

The table below gives a list of CRM library functions.

**Table 98. Summary of CRM library functions**

Function name	Description
crm_reset	Reset clock reset management register and control status
crm_lext_bypass	Configure low-speed external clock bypass
crm_hext_bypass	Configure high-speed external clock bypass
crm_flag_get	Check if the selected flag is set or not
crm_hext_stable_wait	Wait HEXT to get stable
crm_hick_clock_trimming_set	High speed internal clock trimming
crm_hick_clock_calibration_set	High speed internal clock calibration
crm_periph_clock_enable	Peripheral clock enable
crm_periph_reset	Peripheral set
crm_periph_sleep_mode_clock_enable	Peripheral clock enable in Sleep mode
crm_clock_source_enable	Clock source enable
crm_flag_clear	Clear flag

crm_rtc_clock_select	RTC clock source selection
crm_rtc_clock_enable	RTC clock enable
crm_ahb_div_set	AHB clock division
crm_apb1_div_set	APB1 clock division
crm_apb2_div_set	APB2 clock division
crm_adc_clock_div_set	ADC clock division
crm_usb_clock_div_set	USB clock division
crm_clock_failure_detection_enable	Clock failure detection enable
crm_batteryPowered_domain_reset	Battery powered domain reset
crm_pll_config	PLL clock source and frequency multiplication factor
crm_sysclk_switch	System clock source switch
crm_sysclk_switch_status_get	Get the status of system clock source
crm_clocks_freq_get	Get clock frequency
crm_clock_out_set	Clock output clock source
crm_interrupt_enable	Interrupt enable

## 5.5.1 crm\_reset function

The table below describes the function crm\_reset.

**Table 99. crm\_reset function**

Name	Description
Function name	crm_reset
Function prototype	void crm_reset(void);
Function description	Reset the clock reset management register and control status
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

1. This function does not change the HICKTRIM[5:0] bit of the CRM\_CTRL register.
2. Modifying the function does not reset the CRM\_BPDC and CRM\_CTRLSTS registers.

**Example:**

```
/* reset crm */  
crm_reset();
```

## 5.5.2 crm\_lext\_bypass function

The table below describes the function crm\_lext\_bypass.

**Table 100. crm\_lext\_bypass function**

Name	Description
Function name	crm_lext_bypass
Function prototype	void crm_lext_bypass(confirm_state new_state);
Function description	Configure low-speed external clock bypass
Input parameter 1	new_state: Enable bypass (TRUE), disable bypass (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	The LEXT configuration must be done before being enabled.
Called functions	NA

**Example:**

```
/* enable lext bypass mode */  
crm_lext_bypass(TRUE);
```

### 5.5.3 crm\_hext\_bypass function

The table below describes the function crm\_hext\_bypass.

**Table 101. crm\_hext\_bypass function**

Name	Description
Function name	crm_hext_bypass
Function prototype	void crm_hext_bypass(confirm_state new_state);
Function description	Configure high-speed external clock bypass
Input parameter 1	new_state: Enable bypass (TRUE), disable bypass (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	The HEXT configuration must be done before being enabled.
Called functions	NA

**Example:**

```
/* enable hext bypass mode */
crm_hext_bypass(TRUE);
```

### 5.5.4 crm\_flag\_get function

The table below describes the function crm\_flag\_get.

**Table 102. crm\_flag\_get function**

Name	Description
Function name	crm_flag_get
Function prototype	flag_status crm_flag_get(uint32_t flag);
Function description	Check if the selected flag has been set.
Input parameter 1	Flag: flag selection
Input parameter 2	NA
Output parameter	NA
Return value	flag_status: check the status of the selected flag. (SET or RESET)
Required preconditions	NA
Called functions	NA

**flag**

Select a flag to read, including:

CRM_HICK_STABLE_FLAG:	HICK clock stable flag
CRM_HEXT_STABLE_FLAG:	HEXT clock stable flag
CRM_PLL_STABLE_FLAG:	PLL clock stable flag
CRM_LEXT_STABLE_FLAG:	LEXT clock stable flag
CRM_LICK_STABLE_FLAG:	LICK clock stable flag
CRM_NRST_RESET_FLAG:	NRST pin reset flag
CRM_POR_RESET_FLAG:	Power-on/low voltage reset flag
CRM_SW_RESET_FLAG:	Software reset flag
CRM_WDT_RESET_FLAG:	Watchdog reset flag
CRM_WWDT_RESET_FLAG:	Window watchdog reset flag
CRM_LOWPOWER_RESET_FLAG:	Low-power consumption reset flag

CRM_LICK_READY_INT_FLAG:	LICK clock ready interrupt flag
CRM_LEXT_READY_INT_FLAG:	LEXT clock ready interrupt flag
CRM_HICK_READY_INT_FLAG:	HICK clock ready interrupt flag
CRM_HEXT_READY_INT_FLAG:	HEXT clock ready interrupt flag
CRM_PLL_READY_INT_FLAG:	PLL clock ready interrupt flag
CRM_CLOCK_FAILURE_INT_FLAG:	Clock failure interrupt flag

**Example:**

```
/* wait till pll is ready */
while(crm_flag_get(CRM_PLL_STABLE_FLAG) != SET)
{
}
```

### 5.5.5 crm\_interrupt\_flag\_get function

The table below describes the function crm\_interrupt\_flag\_get.

**Table 103. crm\_interrupt\_flag\_get function**

Name	Description
Function name	crm_interrupt_flag_get
Function prototype	flag_status crm_interrupt_flag_get(uint32_t flag);
Function description	Check if the selected flag has been set.
Input parameter 1	flag: flag selection refer to the "flag" description below for details.
Input parameter 2	NA
Output parameter	NA
Return value	flag_status: the status of the selected flag. (SET or RESET)
Required preconditions	NA
Called functions	NA

**flag**

Select a flag to read, including:

CRM_LICK_READY_INT_FLAG:	LICK clock ready interrupt flag
CRM_LEXT_READY_INT_FLAG:	LEXT clock ready interrupt flag
CRM_HICK_READY_INT_FLAG:	HICK clock ready interrupt flag
CRM_HEXT_READY_INT_FLAG:	HEXT clock ready interrupt flag
CRM_PLL_READY_INT_FLAG:	PLL clock ready interrupt flag
CRM_CLOCK_FAILURE_INT_FLAG:	Clock failure interrupt flag

**Example**

```
/* check pll ready interrupt flag */
if(crm_interrupt_flag_get(CRM_PLL_READY_INT_FLAG) != RESET)
{
}
```

## 5.5.6 crm\_hext\_stable\_wait function

The table below describes the function crm\_hext\_stable\_wait.

**Table 104. crm\_hext\_stable\_wait function**

Name	Description
Function name	crm_hext_stable_wait
Function prototype	error_status crm_hext_stable_wait(void);
Function description	Wait for HEXT to activate and become stable
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	error_status: Return the status of HEXT. (SUCCESS or ERROR)
Required preconditions	NA
Called functions	NA

**Example:**

```
/* wait till hext is ready */
while(crm_hext_stable_wait() == ERROR)
{
}
```

## 5.5.7 crm\_hick\_clock\_trimming\_set function

The table below describes the function crm\_hick\_clock\_trimming\_set.

**Table 105. crm\_hick\_clock\_trimming\_set function**

Name	Description
Function name	crm_hick_clock_trimming_set
Function prototype	void crm_hick_clock_trimming_set(uint8_t trim_value);
Function description	Trim HICK clock
Input parameter 1	trim_value: trimming value. Default value is 0x10, configurable range is from 0 to 0x1F.
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* set trimming value */
crm_hick_clock_trimming_set(0x1F);
```

## 5.5.8 crm\_hick\_clock\_calibration\_set function

The table below describes the function crm\_hick\_clock\_calibration\_set.

**Table 106. crm\_hick\_clock\_calibration\_set function**

Name	Description
Function name	crm_hick_clock_calibration_set
Function prototype	void crm_hick_clock_calibration_set(uint8_t cali_value);
Function description	Set HICK clock calibration value
Input parameter 1	cali_value: calibration compensation value. The factory gate value is the default value, Its configurable range is from 0 to 0xFF.
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* set trimming value */
crm_hick_clock_trimming_set(0x80);
```

## 5.5.9 crm\_periph\_clock\_enable function

The table below describes the function crm\_periph\_clock\_enable.

**Table 107. crm\_periph\_clock\_enable function**

Name	Description
Function name	crm_periph_clock_enable
Function prototype	void crm_periph_clock_enable(crm_periph_clock_type value, confirm_state new_state);
Function description	Enable peripheral clock
Input parameter 1	Value: defines peripheral clock type
Input parameter 2	new_state: TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**value**

The crm\_periph\_clock\_type is defined in the at32f403\_crm.h. The naming rule of this parameter is: CRM\_peripheral\_PERIPH\_CLOCK.

CRM\_DMA1\_PERIPH\_CLOCK: DMA1 peripheral clock enable

CRM\_DMA2\_PERIPH\_CLOCK: DMA2 peripheral clock enable

...

CRM\_PWC\_PERIPH\_CLOCK: PWC peripheral clock enable

CRM\_DAC\_PERIPH\_CLOCK: DAC peripheral clock enable

**Example:**

```
/* enable gpioa periph clock */
```

```
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
```

### 5.5.10 crm\_periph\_reset function

The table below describes the function crm\_periph\_reset.

Table 108. crm\_periph\_reset function

Name	Description
Function name	crm_periph_reset
Function prototype	void crm_periph_reset(crm_periph_reset_type value, confirm_state new_state);
Function description	Reset peripherals
Input parameter 1	Value: Peripheral reset type Refer to the “value” description below for details.
Input parameter 2	new_state: TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### value

This indicates the selected peripheral. The crm\_periph\_reset\_type is defined in the at32f403\_crm.h. The naming rule of this parameter is defined as CRM\_peripheral name\_PERIPH\_RESET.

CRM\_DMA1\_PERIPH\_RESET: DMA1 peripheral reset

CRM\_DMA2\_PERIPH\_RESET: DMA 2 peripheral reset

...

CRM\_PWC\_PERIPH\_RESET: PWC peripheral reset

CRM\_DAC\_PERIPH\_RESET: DAC peripheral reset

#### Example:

```
/* reset gpioa periph */  
crm_periph_reset(CRM_GPIOA_PERIPH_RESET, TRUE);
```

## 5.5.11 crm\_periph\_sleep\_mode\_clock\_enable function

The table below describes the function crm\_periph\_sleep\_mode\_clock\_enable.

**Table 109. crm\_periph\_sleep\_mode\_clock\_enable function**

Name	Description
Function name	crm_periph_sleep_mode_clock_enable
Function prototype	void crm_periph_sleep_mode_clock_enable(crm_periph_clock_sleepmd_type value, confirm_state new_state);
Function description	Enable peripheral clock in Sleep mode
Input parameter 1	Value: indicates peripheral clock type in Sleep mode Refer to the “value” description below for details.
Input parameter 2	new_state: TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### value

This indicates the selected peripheral. The crm\_periph\_clock\_sleepmd\_type is defined in the at32f403\_crm.h. The naming rule of this parameter is defined as:

CRM\_peripheral name\_PERIPH\_CLOCK\_SLEEP\_MODE

CRM\_SRAM\_PERIPH\_RESET: SRAM clock reset in Sleep mode

CRM\_FLASH\_PERIPH\_RESET: Flash clock reset in Sleep mode

### Example:

```
/* disable flash clock when entry sleep mode */  
crm_periph_sleep_mode_clock_enable (CRM_FLASH_PERIPH_CLOCK_SLEEP_MODE, FALSE);
```

## 5.5.12 crm\_clock\_source\_enable function

The table below describes the function crm\_clock\_source\_enable.

**Table 110. crm\_clock\_source\_enable function**

Name	Description
Function name	crm_clock_source_enable
Function prototype	void crm_clock_source_enable(crm_clock_source_type source, confirm_state new_state);
Function description	Enable clock source
Input parameter 1	Source: Clock type
Input parameter 2	new_state: TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### source

Clock source selection.

CRM\_CLOCK\_SOURCE\_HICK: HICK  
 CRM\_CLOCK\_SOURCE\_HEXT: HEXT  
 CRM\_CLOCK\_SOURCE\_PLL: PLL  
 CRM\_CLOCK\_SOURCE\_LEXT: LEXT  
 CRM\_CLOCK\_SOURCE\_LICK: LICK

### Example:

```
/* enable hext */
crm_clock_source_enable (CRM_CLOCK_SOURCE_HEXT, FALSE);
```

## 5.5.13 crm\_flag\_clear function

The table below describes the function crm\_flag\_clear.

**Table 111. crm\_flag\_clear function**

Name	Description
Function name	crm_flag_clear
Function prototype	void crm_flag_clear(uint32_t flag);
Function description	Clear the selected flags
Input parameter 1	flag: Indicates the flag to clear Refer to the "flag" description below for details.
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### flag

Select a flag to clear.

CRM_NRST_RESET_FLAG:	NRST pin reset flag
CRM_POR_RESET_FLAG:	Power-on/low voltage reset flag

CRM_SW_RESET_FLAG:	Software reset flag
CRM_WDT_RESET_FLAG:	Watchdog reset flag
CRM_WWDT_RESET_FLAG:	Window watchdog reset flag
CRM_LOWPOWER_RESET_FLAG:	Low-power reset flag
CRM_ALL_RESET_FLAG:	All reset flags
CRM_LICK_READY_INT_FLAG:	LICK clock ready interrupt flag
CRM_LEXT_READY_INT_FLAG:	LEXT clock ready interrupt flag
CRM_HICK_READY_INT_FLAG:	HICK clock ready interrupt flag
CRM_HEXT_READY_INT_FLAG:	HEXT clock ready interrupt flag
CRM_PLL_READY_INT_FLAG:	PLL clock ready interrupt flag
CRM_CLOCK_FAILURE_INT_FLAG:	Clock failure interrupt flag

**Example:**

```
/* clear clock failure detection flag */
crm_flag_clear(CRM_CLOCK_FAILURE_INT_FLAG);
```

### 5.5.14 crm\_RTC\_clock\_select function

The table below describes the function crm\_erc\_c\_clock\_select.

**Table 112. crm\_erc\_c\_clock\_select function**

Name	Description
Function name	crm_RTC_clock_select
Function prototype	void crm_RTC_clock_select(crm_RTC_clock_type value);
Function description	Select RTC clock source
Input parameter 1	Value: indicates erc clock source type
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**value**

ERTC clock source selection.

CRM\_RTC\_CLOCK\_NOCLK: No clock source for RTC

CRM\_RTC\_CLOCK\_LEXT: LEXT selected as RTC clock

CRM\_RTC\_CLOCK\_LICK: LICK selected as RTC clock

CRM\_RTC\_CLOCK\_HEXT\_DIV: HEXT/128 selected as RTC clock

**Example:**

```
/* config lext as rtc clock */
crm_RTC_clock_select (CRM_RTC_CLOCK_LEXT);
```

## 5.5.15 crm\_RTC\_clock\_enable function

The table below describes the function crm\_RTC\_clock\_enable.

**Table 113. crm\_RTC\_clock\_enable function**

Name	Description
Function name	crm_RTC_clock_enable
Function prototype	void crm_RTC_clock_enable(confirm_state new_state);
Function description	Enable RTC clock
Input parameter 1	new_state: TRUE or FALSE
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable rtc clock */
crm_RTC_clock_enable (TRUE);
```

## 5.5.16 crm\_AHB\_div\_set function

The table below describes the function crm\_AHB\_div\_set.

**Table 114. crm\_AHB\_div\_set function**

Name	Description
Function name	crm_AHB_div_set
Function prototype	void crm_AHB_div_set(crm_AHB_div_type value);
Function description	Configure AHB clock division
Input parameter 1	Value: indicates the division factor Refer to the “value” description below for details.
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**value**

- CRM\_AHB\_DIV\_1: SCLK/1 used as AHB clock
- CRM\_AHB\_DIV\_2: SCLK/2 used as AHB clock
- CRM\_AHB\_DIV\_4: SCLK/4 used as AHB clock
- CRM\_AHB\_DIV\_8: SCLK/8 used as AHB clock
- CRM\_AHB\_DIV\_16: SCLK/16 used as AHB clock
- CRM\_AHB\_DIV\_64: SCLK/64 used as AHB clock
- CRM\_AHB\_DIV\_128: SCLK/128 used as AHB clock
- CRM\_AHB\_DIV\_256: SCLK/256 used as AHB clock
- CRM\_AHB\_DIV\_512: SCLK/512 used as AHB clock

**Example:**

```
/* config ahbclk */
```

crm_ahb_div_set(CRM_AHB_DIV_1);
---------------------------------

### 5.5.17 crm\_apb1\_div\_set function

The table below describes the function crm\_apb1\_div\_set.

**Table 115. crm\_apb1\_div\_set function**

Name	Description
Function name	crm_apb1_div_set
Function prototype	void crm_apb1_div_set(crm_apb1_div_type value);
Function description	Configure APB1 clock division
Input parameter 1	Value: indicates the division factor
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**value**

- CRM\_APB1\_DIV\_1: AHB/1 used as APB1 clock
- CRM\_APB1\_DIV\_2: AHB/2 used as APB1 clock
- CRM\_APB1\_DIV\_4: AHB/4 used as APB1 clock
- CRM\_APB1\_DIV\_8: AHB/8 used as APB1 clock
- CRM\_APB1\_DIV\_16: AHB/16 used as APB1 clock

**Example:**

/* config apb1clk */
crm_apb1_div_set(CRM_APB1_DIV_2);

### 5.5.18 crm\_apb2\_div\_set function

The table below describes the function crm\_apb2\_div\_set.

**Table 116. crm\_apb2\_div\_set function**

Name	Description
Function name	crm_apb2_div_set
Function prototype	void crm_apb2_div_set(crm_apb2_div_type value);
Function description	Configure APB2 clock division
Input parameter 1	Value: indicates the division factor
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**value**

- CRM\_APB2\_DIV\_1: AHB/1 used as APB2 clock
- CRM\_APB2\_DIV\_2: AHB/2 used as APB2 clock
- CRM\_APB2\_DIV\_4: AHB/4 used as APB2 clock
- CRM\_APB2\_DIV\_8: AHB/8 used as APB2 clock
- CRM\_APB2\_DIV\_16: AHB/16 used as APB2 clock

**Example:**

```
/* config apb2clk */
crm_apb2_div_set(CRM_APB2_DIV_2);
```

### 5.5.19 crm\_adc\_clock\_div\_set function

The table below describes the function crm\_adc\_clock\_div\_set.

**Table 117. crm\_adc\_clock\_div\_set function**

Name	Description
Function name	crm_adc_clock_div_set
Function prototype	void crm_adc_clock_div_set(crm_adc_div_type div_value);
Function description	Configure ADC clock division
Input parameter 1	div_value: indicate the division factor
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**div\_value**

- CRM\_ADC\_DIV\_2: APB/2 used as ADC clock
- CRM\_ADC\_DIV\_4: APB/4 used as ADC clock
- CRM\_ADC\_DIV\_6: APB/6 used as ADC clock
- CRM\_ADC\_DIV\_8: APB/8 used as ADC clock
- CRM\_ADC\_DIV\_12: APB/12 used as ADC clock
- CRM\_ADC\_DIV\_16: APB/16 used as ADC clock

**Example:**

```
/* config adc div 4 */
crm_adc_clock_div_set(CRM_ADC_DIV_4);
```

### 5.5.20 crm\_usb\_clock\_div\_set function

The table below describes the function crm\_usb\_clock\_div\_set.

**Table 118. crm\_usb\_clock\_div\_set function**

Name	Description
Function name	crm_usb_clock_div_set
Function prototype	void crm_usb_clock_div_set(crm_usb_div_type div_value);
Function description	Configure USB clock division
Input parameter 1	div_value: division factor Refer to the "div_value" descriptions below for details.
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**div\_value**

- CRM\_USB\_DIV\_1\_5: PLL/1.5 as USB clock
- CRM\_USB\_DIV\_1: PLL/1 as USB clock
- CRM\_USB\_DIV\_2\_5: PLL/2.5 as USB clock
- CRM\_USB\_DIV\_2: PLL/2 as USB clock
- CRM\_USB\_DIV\_3\_5: PLL/3.5 as USB clock
- CRM\_USB\_DIV\_3: PLL/3 as USB clock
- CRM\_USB\_DIV\_4: PLL/4 as USB clock

**Example:**

```
/* config usb div 2 */
crm_usb_clock_div_set(CRM_USB_DIV_2);
```

### 5.5.21 **crm\_clock\_failure\_detection\_enable** function

The table below describes the function `crm_clock_failure_detection_enable`.

**Table 119. `crm_clock_failure_detection_enable` function**

Name	Description
Function name	<code>crm_clock_failure_detection_enable</code>
Function prototype	<code>void crm_clock_failure_detection_enable(confirm_state new_state);</code>
Function description	Enable clock failure detection
Input parameter 1	<code>new_state</code> : TRUE or FALSE
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable clock failure detection */
crm_clock_failure_detection_enable(TRUE);
```

### 5.5.22 **crm\_batteryPoweredDomainReset** function

The table below describes the function `crm_batteryPoweredDomainReset`.

**Table 120. `crm_batteryPoweredDomainReset` function**

Name	Description
Function name	<code>crm_batteryPoweredDomainReset</code>
Function prototype	<code>void crm_batteryPoweredDomainReset(confirm_state new_state);</code>
Function description	Reset battery powered domain
Input parameter 1	<code>new_state</code> : Reset (TRUE), Not reset (FALSE)
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

When it comes to resetting battery powered domain, it is usually necessary to reset battery powered domain through TRUE operation and then disable battery powered domain reset through FALSE operation after the completion of reset.

**Example:**

```
/* reset battery powered domain */
crm_batteryPoweredDomainReset (TRUE);
```

### 5.5.23 crm\_pll\_config function

The table below describes the function crm\_pll\_config.

**Table 121. crm\_pll\_config function**

Name	Description
Function name	crm_pll_config
Function prototype	void crm_pll_config(crm_pll_clock_source_type clock_source, crm_pll_mult_type mult_value, crm_pll_output_range_type pll_range);
Function description	Configure PLL clock source and frequency multiplication factor
Input parameter 1	clock_source: clock source for PLL frequency multiplication Refer to the “clock_source” descriptions below for details.
Input parameter 2	mult_value: frequency multiplication factor Refer to the “mult_value” descriptions below for details.
Input parameter 3	pll_range: set it depending on whether PLL output range is greater than 72 MHz or not
Output parameter	NA
Return value	NA
Required preconditions	PLL clock source must be enabled and stable before enabling PLL
Called functions	NA

**clock\_source**

CRM\_PLL\_SOURCE\_HICK: HICK as PLL clock

CRM\_PLL\_SOURCE\_HEXT: HEXT as PLL clock

CRM\_PLL\_SOURCE\_HEXT\_DIV: Divided HEXT as PLL clock

**mult\_value**

CRM\_PLL\_MULT\_2: PLL input clock x2

CRM\_PLL\_MULT\_3: PLL input clock x3

...

CRM\_PLL\_MULT\_63: PLL input clock x63

CRM\_PLL\_MULT\_64: PLL input clock x64

**pll\_range**

CRM\_PLL\_OUTPUT\_RANGE\_LE72MHZ: PLL output frequency is lower than 72MHz

CRM\_PLL\_OUTPUT\_RANGE\_GT72MHZ: PLL output frequency is greater than 72MHz

**Example:**

```
/* config pll clock resource */
crm_pllConfig(CRM_PLL_SOURCE_HEXT_DIV, CRM_PLL_MULT_60,
CRM_PLL_OUTPUT_RANGE_GT72MHZ);
```

## 5.5.24 crm\_sysclk\_switch function

The table below describes the function crm\_sysclk\_switch.

**Table 122. crm\_sysclk\_switch function**

Name	Description
Function name	crm_sysclk_switch
Function prototype	void crm_sysclk_switch(crm_sclk_type value);
Function description	Switch system clock source
Input parameter 1	Value: indicates the clock source for system clock Refer to the “value” descriptions below for details.
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**value**

- CRM\_SCLK\_HICK: HICK as system clock
- CRM\_SCLK\_HEXT: HEXT as system clock
- CRM\_SCLK\_PLL: PLL as system clock

**Example:**

```
/* select pll as system clock source */
crm_sysclk_switch(CRM_SCLK_PLL);
```

## 5.5.25 crm\_sysclk\_switch\_status\_get function

The table below describes the function crm\_sysclk\_switch\_status\_get.

**Table 123. crm\_sysclk\_switch\_status\_get function**

Name	Description
Function name	crm_sysclk_switch_status_get
Function prototype	crm_sclk_type crm_sysclk_switch_status_get(void);
Function description	Get the clock source of System clock
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	crm_sclk_type: return value is the clock source of system clock
Required preconditions	NA
Called functions	NA

**Example:**

```
/* wait till pll is used as system clock source */
while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL)
{
}
```

## 5.5.26 crm\_clocks\_freq\_get function

The table below describes the function crm\_clocks\_freq\_get.

**Table 124. crm\_clocks\_freq\_get function**

Name	Description
Function name	crm_clocks_freq_get
Function prototype	void crm_clocks_freq_get(crm_clocks_freq_type *clocks_struct);
Function description	Get clock frequency
Input parameter 1	clocks_struct: crm_clocks_freq_type pointer, including clock frequency Refer to the “crm_clocks_freq_type” for details.
Input parameter 2	NA
Output parameter	NA
Return value	crm_sclk_type: return the clock source for system clock
Required preconditions	NA
Called functions	NA

### crm\_clocks\_freq\_type

The crm\_clocks\_freq\_type is defined in the at32f403\_crm.h:

typedef struct

```
{
    uint32_t    sclk_freq;
    uint32_t    ahb_freq;
    uint32_t    apb2_freq;
    uint32_t    apb1_freq;
    uint32_t    adc_freq;
}
```

} crm\_clocks\_freq\_type;

### sclk\_freq

Get the system clock frequency, in Hz

### ahb\_freq

Get the clock frequency of AHB, in Hz

### apb2\_freq

Get the clock frequency of APB2, in Hz

### apb1\_freq

Get the clock frequency of APB1, in Hz

### adc\_freq

Get the clock frequency of ADC, in Hz

### Example:

```
/* get frequency */
crm_clocks_freq_type clocks_struct;
crm_clocks_freq_get(&clocks_struct);
```

## 5.5.27 crm\_clock\_out\_set function

The table below describes the function crm\_clock\_out\_set.

**Table 125. crm\_clock\_out\_set function**

Name	Description
Function name	crm_clock_out_set
Function prototype	void crm_clock_out_set(crm_clkout_select_type clkout);
Function description	Configure clock output source on the clkout pin
Input parameter 1	Clkout: the clock output source on the clkout pin
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* config PA8 output pll/4 */
crm_clock_out_set(CRM_CLKOUT_PLL_DIV_4);
```

## 5.5.28 crm\_interrupt\_enable function

The table below describes the function crm\_interrupt\_enable.

**Table 126. crm\_interrupt\_enable function**

Name	Description
Function name	crm_interrupt_enable
Function prototype	void crm_interrupt_enable(uint32_t crm_int, confirm_state new_state);
Function description	Enable interrupts
Input parameter 1	crm_int: indicates the selected interrupt Refer to the "crm_int" for details.
Input parameter 2	new_state: Enable (TRUE), disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**crm\_int**

CRM_LICK_STABLE_INT:	LICK stable interrupt
CRM_LEXT_STABLE_INT:	LEXT stable interrupt
CRM_HICK_STABLE_INT:	HICK stable interrupt
CRM_HEXT_STABLE_INT:	HEXT stable interrupt
CRM_PLL_STABLE_INT:	PLL stable interrupt
CRM_CLOCK_FAILURE_INT:	Clock failure interrupt

**Example:**

```
/* enable pll stable interrupt */
crm_interrupt_enable (CRM_PLL_STABLE_INT);
```

## 5.6 Digital-to-analog converter (DAC)

The DAC register structure `dac_type` is defined in the “`at32f403_dac.h`”:

```
/*
 * @brief type define dac register all
 */
typedef struct
{
    ...
} dac_type;
```

The table below gives a list of the DAC registers.

**Table 127. Summary of DAC registers**

Register	Description
ctrl	DAC control register
swtrg	DAC software trigger register
d1dth12r	DAC1 12-bit right-aligned data holding register
d1dth12l	DAC1 12-bit left-aligned data holding register
d1dth8r	DAC1 8-bit right-aligned data holding register
d2dth12r	DAC2 12-bit right-aligned data holding register
d2dth12l	DAC2 12-bit left-aligned data holding register
d2dth8r	DAC2 8-bit right-aligned data holding register
ddth12r	Dual DAC 12-bit right-aligned data holding register
ddth12l	Dual DAC 12-bit left-aligned data holding register
ddth8r	Dual DAC 8-bit right-aligned data holding register
d1odt	DAC1 data output register
d2odt	DAC2 data output register

The table below gives a list of DAC library functions.

**Table 128. Summary of DAC library functions**

Function name	Description
dac_reset	Reset all DAC registers to their reset values
dac_enable	Enable DAC
dac_output_buffer_enable	Enable DAC output buffer
dac_trigger_enable	Enable DAC trigger
dac_trigger_select	Select DAC trigger source
dac_software_trigger_generate	Trigger DAC by software
dac_dual_software_trigger_generate	Simultaneous trigger DAC1 and DAC2 by software
dac_wave_generate	Select DAC output waveform
dac_mask_amplitude_select	Select DAC noise /triangle-wave amplitude
dac_dma_enable	DAC DMA enable
dac_data_output_get	Get DAC output value
dac_1_data_set	Set DAC1 output value
dac_2_data_set	Set DAC2 output value
dac_dual_data_set	Set DAC1 and DAC2 output values

## 5.6.1 dac\_reset function

The table below describes the function dac\_reset.

**Table 129. dac\_reset function**

Name	Description
Function name	dac_reset
Function prototype	void dac_reset(void);
Function description	Reset all DAC registers to their reset values
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

**Example:**

```
dac_reset();
```

## 5.6.2 dac\_enable function

The table below describes the function dac\_enable.

**Table 130. dac\_enable function**

Name	Description
Function name	dac_enable
Function prototype	void dac_enable(dac_select_type dac_select, confirm_state new_state);
Function description	Enable DAC
Input parameter 1	dac_select: Select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
dac_enable(DAC1_SELECT, TRUE);
```

### 5.6.3 dac\_output\_buffer\_enable function

The table below describes the function dac\_output\_buffer\_enable.

**Table 131. dac\_output\_buffer\_enable function**

Name	Description
Function name	dac_output_buffer_enable
Function prototype	void dac_output_buffer_enable(dac_select_type dac_select, confirm_state new_state);
Function description	Enable DAC output buffer
Input parameter 1	dac_select: Select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
dac_output_buffer_enable (DAC1_SELECT, TRUE);
```

### 5.6.4 dac\_trigger\_enable function

The table below describes the function dac\_trigger\_enable.

**Table 132. dac\_trigger\_enable function**

Name	Description
Function name	dac_trigger_enable
Function prototype	void dac_trigger_enable(dac_select_type dac_select, confirm_state new_state);
Function description	Enable DAC trigger
Input parameter 1	dac_select: Select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
dac_trigger_enable (DAC1_SELECT, TRUE);
```

## 5.6.5 dac\_trigger\_select function

The table below describes the function dac\_trigger\_select.

Table 133. dac\_trigger\_select function

Name	Description
Function name	dac_trigger_select
Function prototype	void dac_trigger_select(dac_select_type dac_select, dac_trigger_type dac_trigger_source);
Function description	Select DAC trigger source
Input parameter 1	dac_select: Select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	<i>dac_trigger_source</i> : trigger source selected
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### dac\_trigger\_source

Trigger source selection.

DAC_TMR6_TRGOUT_EVENT:	TMR6 TRGOUT event triggers DAC
DAC_TMR8_TRGOUT_EVENT:	TMR8 TRGOUT event triggers DAC
DAC_TMR7_TRGOUT_EVENT:	TMR7 TRGOUT event triggers DAC
DAC_TMR5_TRGOUT_EVENT:	TMR5 TRGOUT event triggers DAC
DAC_TMR2_TRGOUT_EVENT:	TMR2 TRGOUT event triggers DAC
DAC_TMR4_TRGOUT_EVENT:	TMR4 TRGOUT event triggers DAC
DAC_EXTERNAL_INTERRUPT_LINE_9:	EXINT LINE 9 event triggers DAC
DAC_SOFTWARE_TRIGGER:	Software triggers DAC

### Example:

```
dac_trigger_select(DAC1_SELECT, DAC_TMR2_TRGOUT_EVENT);
dac_trigger_select(DAC2_SELECT, DAC_TMR2_TRGOUT_EVENT);
```

## 5.6.6 dac\_software\_trigger\_generate function

The table below describes the function dac\_software\_trigger\_generate.

**Table 134. dac\_software\_trigger\_generate function**

Name	Description
Function name	dac_software_trigger_generate
Function prototype	void dac_software_trigger_generate(dac_select_type dac_select);
Function description	Trigger DAC by software
Input parameter 1	dac_select: Select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
dac_software_trigger_generate (DAC1_SELECT);
```

## 5.6.7 dac\_dual\_software\_trigger\_generate function

The table below describes the function dac\_dual\_software\_trigger\_generate.

**Table 135. dac\_dual\_software\_trigger\_generate function**

Name	Description
Function name	dac_dual_software_trigger_generate
Function prototype	void dac_dual_software_trigger_generate(void);
Function description	Trigger DAC1 and DAC2 by software simultaneously
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
dac_dual_software_trigger_generate ();
```

## 5.6.8 dac\_wave\_generate function

The table below describes the function dac\_wave\_generate.

**Table 136. dac\_wave\_generate function**

Name	Description
Function name	dac_wave_generate
Function prototype	void dac_wave_generate(dac_select_type dac_select, dac_wave_type dac_wave);
Function description	DAC wave generation enable
Input parameter 1	dac_select: Select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	<i>dac_wave</i> : wave selected
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### dac\_wave

Wave generation enable.

DAC\_WAVE\_GENERATE\_NONE: Wave generation disabled (output fixed register value)

DAC\_WAVE\_GENERATE\_NOISE: Noise wave generation

DAC\_WAVE\_GENERATE\_TRIANGLE: Triangle wave generation

### Example:

```
dac_wave_generate(DAC1_SELECT, DAC_WAVE_GENERATE_NONE);
dac_wave_generate(DAC2_SELECT, DAC_WAVE_GENERATE_NOISE);
```

## 5.6.9 dac\_mask\_amplitude\_select function

The table below describes the function dac\_mask\_amplitude\_select.

**Table 137. dac\_mask\_amplitude\_select function**

Name	Description
Function name	dac_mask_amplitude_select
Function prototype	void dac_mask_amplitude_select(dac_select_type dac_select, dac_mask_amplitude_type dac_mask_amplitude);
Function description	DAC noise bit width/triangle amplitude selection
Input parameter 1	dac_select: Select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	<i>dac_mask_amplitude</i> : noise bit width/triangle amplitude selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### dac\_mask\_amplitude

Noise bit width/triangle amplitude selection.

DAC\_LSFR\_BIT0\_AMPLITUDE\_1:

LSFR[0] in noise mode/triangle amplitude equal to 1

DAC\_LSFR\_BIT10\_AMPLITUDE\_3:  
 LSFR[1:0] in noise mode/triangle amplitude equal to 3

DAC\_LSFR\_BIT20\_AMPLITUDE\_7:  
 LSFR[2:0] in noise mode/triangle amplitude equal to 7

DAC\_LSFR\_BIT30\_AMPLITUDE\_15:  
 LSFR[3:0] in noise mode/triangle amplitude equal to 15

DAC\_LSFR\_BIT40\_AMPLITUDE\_31:  
 LSFR[4:0] in noise mode/triangle amplitude equal to 31

DAC\_LSFR\_BIT50\_AMPLITUDE\_63:  
 LSFR[5:0] in noise mode/triangle amplitude equal to 63

DAC\_LSFR\_BIT60\_AMPLITUDE\_127:  
 LSFR[6:0] in noise mode/triangle amplitude equal to 127

DAC\_LSFR\_BIT70\_AMPLITUDE\_255:  
 LSFR[7:0] in noise mode/triangle amplitude equal to 255

DAC\_LSFR\_BIT80\_AMPLITUDE\_511:  
 LSFR[8:0] in noise mode/triangle amplitude equal to 511

DAC\_LSFR\_BIT90\_AMPLITUDE\_1023:  
 LSFR[9:0] in noise mode/triangle amplitude equal to 1023

DAC\_LSFR\_BITA0\_AMPLITUDE\_2047:  
 LSFR[10:0] in noise mode/triangle amplitude equal to 2047

DAC\_LSFR\_BITB0\_AMPLITUDE\_4095:  
 LSFR[11:0] in noise mode/triangle amplitude equal to 4095

**Example:**

```
dac_mask_amplitude_select (DAC1_SELECT, DAC_LSFR_BIT60_AMPLITUDE_127);
dac_mask_amplitude_select (DAC2_SELECT, DAC_LSFR_BIT80_AMPLITUDE_511);
```

### 5.6.10 dac\_dma\_enable function

The table below describes the function dac\_dma\_enable.

**Table 138. dac\_dma\_enable function**

Name	Description
Function name	dac_dma_enable
Function prototype	void dac_dma_enable(dac_select_type dac_select, confirm_state new_state);
Function description	DAC DMA enable
Input parameter 1	dac_select: Select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
dac_dma_enable (DAC1_SELECT, TRUE);
```

## 5.6.11 dac\_data\_output\_get function

The table below describes the function dac\_data\_output\_get.

**Table 139. dac\_data\_output\_get function**

Name	Description
Function name	dac_data_output_get
Function prototype	uint16_t dac_data_output_get(dac_select_type dac_select);
Function description	Get DAC output value
Input parameter 1	dac_select: Select a DAC This parameter can be DAC1_SELECT or DAC2_SELECT.
Output parameter	NA
Return value	dacx_data: dac1/dac2 output value
Required preconditions	NA
Called functions	NA

**Example:**

```
uint16_t dac1_data;
dac1_data = dac_data_output_get (DAC1_SELECT);
```

## 5.6.12 dac\_1\_data\_set function

The table below describes the function dac\_1\_data\_set.

**Table 140. dac\_1\_data\_set function**

Name	Description
Function name	dac_1_data_set
Function prototype	void dac_1_data_set(dac1_aligned_data_type dac1_aligned, uint16_t dac1_data);
Function description	Set DAC1 output data format and range
Input parameter 1	<i>dac1_aligned</i> : data format selection
Input parameter 2	<i>dac1_data</i> : DAC output value
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### dac1\_aligned

Data format selector.

DAC1\_12BIT\_RIGHT: 12-bit right-aligned

DAC1\_12BIT\_LEFT: 12-bit left-aligned

DAC1\_8BIT\_RIGHT: 8-bit right-aligned

### dac1\_data

DAC output range settings. Value range varies from one format to another.

DAC1\_12BIT\_RIGHT: 0x000~0xFFFF

DAC1\_12BIT\_LEFT: 0x0000~0xFFFF

DAC1\_8BIT\_RIGHT: 0x00~0xFF

**Example:**

```
dac_1_data_set (DAC1_12BIT_RIGHT, 0x666);
```

## 5.6.13 dac\_2\_data\_set function

The table below describes the function dac\_2\_data\_set.

**Table 141. dac\_2\_data\_set function**

Name	Description
Function name	dac_2_data_set
Function prototype	void dac_2_data_set(dac2_aligned_data_type dac2_aligned, uint16_t dac2_data);
Function description	Set DAC2 output data format and range
Input parameter 1	<i>dac2_aligned</i> : data format selection
Input parameter 2	<i>dac2_data</i> : DAC output value
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### **dac2\_aligned**

Data format selector.

DAC2\_12BIT\_RIGHT: 12-bit right-aligned

DAC2\_12BIT\_LEFT: 12-bit left-aligned

DAC2\_8BIT\_RIGHT: 8-bit right-aligned

### **dac2\_data**

DAC output range settings. Value range varies from one format to another.

DAC2\_12BIT\_RIGHT: 0x000~0xFFF

DAC2\_12BIT\_LEFT: 0x0000~0xFFFF

DAC2\_8BIT\_RIGHT: 0x00~0xFF

### **Example:**

```
dac_2_data_set (DAC2_12BIT_RIGHT, 0x666);
```

## 5.6.14 dac\_dual\_data\_set function

The table below describes the function dac\_dual\_data\_set

**Table 142. dac\_dual\_data\_set function**

Name	Description
Function name	dac_dual_data_set
Function prototype	void dac_dual_data_set(dac_dual_data_type dac_dual, uint16_t data1, uint16_t data2);
Function description	DAC1/DAC2 output data format and range
Input parameter 1	<i>dac_dual</i> : data format selection
Input parameter 2	<i>data1</i> : DAC1 output value
Input parameter 3	<i>data2</i> : DAC2 output value
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### **dac\_dual**

Data format selector.

DAC\_DUAL\_12BIT\_RIGHT: 12-bit right-aligned

DAC\_DUAL\_12BIT\_LEFT: 12-bit left-aligned

DAC\_DUAL\_8BIT\_RIGHT: 8-bit right-aligned

### **data1/data2**

DAC output range settings. Value range varies from one format to another.

DAC2\_12BIT\_RIGHT: 0x000~0xFFFF

DAC2\_12BIT\_LEFT: 0x0000~0xFFFF0

DAC2\_8BIT\_RIGHT: 0x00~0xFF

### **Example:**

```
dac_dual_data_set (DAC2_12BIT_RIGHT, 0x666, 0x777);
```

## 5.7 Debug

The DEBUG register structure debug\_type is defined in the “at32f403\_debug.h”:

```
/*
 * @brief type define debug register all
 */
typedef struct
{
    ...
} debug_type;
```

The table below gives a list of the DEBUG registers.

**Table 143. Summary of debug registers**

Register	Description
idcode	Device ID
ctrl	Control register

The table below gives a list of DEBUG library functions.

**Table 144. Summary of debug library functions**

Function name	Description
debug_device_id_get	Read device idcode
debug_periph_mode_set	Peripheral debug mode configuration

### 5.7.1 debug\_device\_id\_get function

The table below describes the function debug\_device\_id\_get

**Table 145. debug\_device\_id\_get function**

Name	Description
Function name	debug_device_id_get
Function prototype	uint32_t debug_device_id_get(void);
Function description	Read device ID code
Input parameter 1	NA
Input parameter 2	NA
Output parameter	NA
Return value	Return 32-bit ID code
Required preconditions	NA
Called functions	NA

**Example:**

```
/* get idcode */
uint32_t idcode = 0;
idcode = debug_device_id_get();
```

## 5.7.2 debug\_periph\_mode\_set function

The table below describes the function debug\_periph\_mode\_set

**Table 146. debug\_periph\_mode\_set function**

Name	Description
Function name	debug_periph_mode_set
Function prototype	void debug_periph_mode_set(uint32_t periph_debug_mode, confirm_state new_state);
Function description	Select a peripheral or mode to debug
Input parameter 1	periph_debug_mode: Select a peripheral or mode
Input parameter 2	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### periph\_debug\_mode

Select a peripheral or mode to debug

DEBUG_SLEEP:	Debug in Sleep mode
DEBUG_DEEPSLEEP:	Debug in Deepsleep mode
DEBUG_STANDBY:	Debug in Standby mode
DEBUG_WDT_PAUSE:	Watchdog pause control bit
DEBUG_WWDT_PAUSE:	Window watchdog pause control bit
DEBUG_TMR1_PAUSE:	TMR1 pause control bit
DEBUG_TMR2_PAUSE:	TMR2 pause control bit
DEBUG_TMR3_PAUSE:	TMR3 pause control bit
DEBUG_TMR4_PAUSE:	TMR4 pause control bit
DEBUG_TMR5_PAUSE:	TMR5 pause control bit
DEBUG_TMR6_PAUSE:	TMR6 pause control bit
DEBUG_TMR7_PAUSE:	TMR7 pause control bit
DEBUG_TMR8_PAUSE:	TMR8 pause control bit
DEBUG_TMR9_PAUSE:	TMR9 pause control bit
DEBUG_TMR10_PAUSE:	TMR10 pause control bit
DEBUG_TMR11_PAUSE:	TMR11 pause control bit
DEBUG_TMR12_PAUSE:	TMR12 pause control bit
DEBUG_TMR13_PAUSE:	TMR13 pause control bit
DEBUG_TMR14_PAUSE:	TMR14 pause control bit
DEBUG_TMR15_PAUSE:	TMR15 pause control bit
DEBUG_I2C1_SMBUS_TIMEOUT:	I2C1 SMBUS TIMEOUT pause control bit
DEBUG_I2C2_SMBUS_TIMEOUT:	I2C2 SMBUS TIMEOUT pause control bit
DEBUG_I2C3_SMBUS_TIMEOUT:	I2C3 SMBUS TIMEOUT pause control bit
DEBUG_CAN1_PAUSE:	CAN1 receive register pause control bit

### Example:

```
/* enable tmr1 debug mode */
debug_periph_mode_set(DEBUG_TMR1_PAUSE, TRUE);
```

## 5.8 DMA controller

The DMA register structure `dma_type` is defined in the “`at32f403_dma.h`”:

```
/*
 * @brief type define dma register
 */
typedef struct
{
    ...
} dma_type;
```

DMA channel register structure `dma_channel_type` is defined in the “`at32f403_dma.h`”:

```
/*
 * @brief type define dma channel register all
 */
typedef struct
{
    ...
} dma_channel_type;
```

The table below gives a list of the DMA registers.

**Table 147. Summary of DMA registers**

Register	Description
<code>dma_sts</code>	DMA status register
<code>dma_clr</code>	DMA status clear register
<code>dma_c1ctrl</code>	DMA channel 1 configuration register
<code>dma_c1dtcnt</code>	DMA channel 1 number of data register
<code>dma_c1paddr</code>	DMA channel 1 peripheral address register
<code>dma_c1maddr</code>	DMA channel 1 memory address register
<code>dma_c2ctrl</code>	DMA channel 2 configuration register
<code>dma_c2dtcnt</code>	DMA channel 2 number of data register
<code>dma_c2paddr</code>	DMA channel 2 peripheral address register
<code>dma_c2maddr</code>	DMA channel 2 memory address register
<code>dma_c3ctrl</code>	DMA channel 3 configuration register
<code>dma_c3dtcnt</code>	DMA channel 3 number of data register
<code>dma_c3paddr</code>	DMA channel 3 peripheral address register
<code>dma_c3maddr</code>	DMA channel 3 memory address register
<code>dma_c4ctrl</code>	DMA channel 4 configuration register
<code>dma_c4dtcnt</code>	DMA channel 4 number of data register
<code>dma_c4paddr</code>	DMA channel 4 peripheral address register
<code>dma_c4maddr</code>	DMA channel 4 memory address register
<code>dma_c5ctrl</code>	DMA channel 5 configuration register
<code>dma_c5dtcnt</code>	DMA channel 5 number of data register

Register	Description
dma_c5paddr	DMA channel 5 peripheral address register
dma_c5maddr	DMA channel 5 memory address register
dma_c6ctrl	DMA channel 6 configuration register
dma_c6dtcnt	DMA channel 6 number of data register
dma_c6paddr	DMA channel 6 peripheral address register
dma_c6maddr	DMA channel 6 memory address register
dma_c7ctrl	DMA channel 7 configuration register
dma_c7dtcnt	DMA channel 7 number of data register
dma_c7paddr	DMA channel 7 peripheral address register
dma_c7maddr	DMA channel 7 memory address register

The table below gives a list of DMA library functions.

**Table 148. Summary of DMA library functions**

Function name	Description
dma_default_para_init	Initialize the parameters of the dma_init_struct
dma_init	Initialize the selected channel
dma_reset	Reset the selected DMA channel
dma_data_number_set	Set the number of data transfer of a given channel
dma_data_number_get	Get the number of data transfer of a given channel
dma_interrupt_enable	Enable DMA channel interrupt
dma_channel_enable	Enable DMA channel
dma_flexible_config	Configure flexible DMA request mapping
dma_flag_get	Get the flag of DMA channels
dma_flag_clear	Clear the flag of DMA channels

## 5.8.1 dma\_default\_para\_init function

The table below describes the function dma\_default\_para\_init

**Table 149. dma\_default\_para\_init function**

Name	Description
Function name	dma_default_para_init
Function prototype	void dma_default_para_init(dma_init_type* dma_init_struct);
Function description	Initialize the parameters of the dma_init_struct
Input parameter 1	dma_init_struct: dma_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of the dma\_init\_struct members.

**Table 150.dma\_init\_struct default values**

Member	Default values
peripheral_base_addr	0x0
memory_base_addr	0x0
direction	DMA_DIR_PERIPHERAL_TO_MEMORY
buffer_size	0x0
peripheral_inc_enable	FALSE
memory_inc_enable	FALSE
peripheral_data_width	DMA_PERIPHERAL_DATA_WIDTH_BYTE
memory_data_width	DMA_MEMORY_DATA_WIDTH_BYTE
loop_mode_enable	FALSE
priority	DMA_PRIORITY_LOW

**Example:**

```
/* dma init config with its default value */
dma_init_type dma_init_struct = {0};
dma_default_para_init(&dma_init_struct);
```

## 5.8.2 dma\_init function

The table below describes the function dma\_init

**Table 151. dma\_init function**

Name	Description
Function name	dma_init
Function prototype	void dma_init(dma_channel_type* dmax_channely, dma_init_type* dma_init_struct)
Function description	Initialize the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Input parameter 2	dma_init_struct: dma_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### dma\_init\_type structure

The dma\_init\_type is defined in the at32f403\_dma.h:

typedef struct

```
{
    uint32_t peripheral_base_addr;
    uint32_t memory_base_addr;
    dma_dir_type direction;
    uint16_t buffer_size;
    confirm_state peripheral_inc_enable;
    confirm_state memory_inc_enable;
    dma_peripheral_data_size_type peripheral_data_width;
    dma_memory_data_size_type memory_data_width;
    confirm_state loop_mode_enable;
    dma_priority_level_type priority;
} dma_init_type;
```

#### peripheral\_base\_addr

Set the peripheral address of a DMA channel

#### memory\_base\_addr

Set the memory address of a DMA channel

#### direction

Set the transfer direction of a DMA channel

DMA\_DIR\_PERIPHERAL\_TO\_MEMORY: Peripheral to memory

DMA\_DIR\_MEMORY\_TO\_PERIPHERAL: Memory to peripheral

DMA\_DIR\_MEMORY\_TO\_MEMORY: Memory to memory

#### buffer\_size

Set the number of data transfer of a DMA channel

#### peripheral\_inc\_enable

Enable/disable DMA channel peripheral address auto increment

FALSE: Peripheral address is not incremented

TRUE: Peripheral address is incremented

**memory\_inc\_enable**

Enable/disable DMA channel memory address auto increment

FALSE: Memory address is not incremented

TRUE: Memory address is incremented

**peripheral\_data\_width**

Set DMA peripheral data width

DMA\_PERIPHERAL\_DATA\_WIDTH\_BYTE: Byte

DMA\_PERIPHERAL\_DATA\_WIDTH\_HALFWORD: Half-word

DMA\_PERIPHERAL\_DATA\_WIDTH\_WORD: Word

**memory\_data\_width**

Set DMA memory data width

DMA\_MEMORY\_DATA\_WIDTH\_BYTE: Byte

DMA\_MEMORY\_DATA\_WIDTH\_HALFWORD: Half-word

DMA\_MEMORY\_DATA\_WIDTH\_WORD: Word

**loop\_mode\_enable**

Set DMA loop mode

FALSE: DMA single mode

TRUE: DMA loop mode

**priority**

Set DMA channel priority

DMA\_PRIORITY\_LOW: Low

DMA\_PRIORITY\_MEDIUM: Medium

DMA\_PRIORITY\_HIGH: High

DMA\_PRIORITY VERY HIGH: Very high

**Example:**

```
dma_init_type dma_init_struct = {0};  
/* dma2 channel1 configuration */  
dma_init_struct.buffer_size = BUFFER_SIZE;  
dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;  
dma_init_struct.memory_base_addr = (uint32_t)src_buffer;  
dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;  
dma_init_struct.memory_inc_enable = TRUE;  
dma_init_struct.peripheral_base_addr = (uint32_t)0x4001100C;  
dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;  
dma_init_struct.peripheral_inc_enable = FALSE;  
dma_init_struct.priority = DMA_PRIORITY_MEDIUM;  
dma_init_struct.loop_mode_enable = FALSE;  
dma_init(DMA2_CHANNEL1, &dma_init_struct);
```

### 5.8.3 dma\_reset function

The table below describes the function dma\_reset

**Table 152. dma\_reset function**

Name	Description
Function name	dma_reset
Function prototype	void dma_reset(dma_channel_type* dmax_channel);
Function description	Reset the selected DMA channel
Input parameter 1	dmax_channel: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* reset dma2 channel1 */
dma_reset(DMA2_CHANNEL1);
```

### 5.8.4 dma\_data\_number\_set function

The table below describes the function dma\_data\_number\_set

**Table 153. dma\_data\_number\_set function**

Name	Description
Function name	dma_data_number_set
Function prototype	void dma_data_number_set(dma_channel_type* dmax_channel, uint16_t data_number);
Function description	Set the number of data transfer of the selected DMA channel
Input parameter 1	dmax_channel: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Input parameter 2	data_number: indicates the number of data transfer, up to 65535
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* set dma2 channel1 data count is 0x100*/
dma_data_number_set(DMA2_CHANNEL1, 0x100);
```

## 5.8.5 dma\_data\_number\_get function

The table below describes the function dma\_data\_number\_get

**Table 154. dma\_data\_number\_get function**

Name	Description
Function name	dma_data_number_get
Function prototype	uint16_t dma_data_number_get(dma_channel_type* dmax_channely);
Function description	Get the number of data transfer of the selected DMA channel
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Output parameter	NA
Return value	Get the number of data transfer of a DMA channel
Required preconditions	NA
Called functions	NA

**Example:**

```
/* get dma2 channel1 data count*/
uint16_t data_counter;
data_counter = dma_data_number_get(DMA2_CHANNEL1);
```

## 5.8.6 dma\_interrupt\_enable function

The table below describes the function dma\_interrupt\_enable

**Table 155. dma\_interrupt\_enable function**

Name	Description
Function name	dma_interrupt_enable
Function prototype	void dma_interrupt_enable(dma_channel_type* dmax_channely, uint32_t dma_int, confirm_state new_state);
Function description	Enable DMA channels interrupt
Input parameter 1	dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Input parameter 2	dma_int: interrupt source selection
Input parameter 3	new_state: interrupt enable/disable
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**dma\_int**

Select DMA interrupt source

DMA\_FDT\_INT: Transfer complete interrupt

DMA\_HDT\_INT: Half transfer complete interrupt

DMA\_DTERR\_INT: Transfer error interrupt

**new\_state**

Enable or disable DMA channel interrupt

FALSE: Disabled

TRUE: Enabled

**Example:**

```
/* enable dma2 channel1 transfer full data interrupt */
dma_interrupt_enable(DMA2_CHANNEL1, DMA_FDT_INT, TRUE);
```

## 5.8.7 dma\_channel\_enable function

The table below describes the function `dma_channel_enable`

**Table 156. `dma_channel_enable` function**

Name	Description
Function name	<code>dma_channel_enable</code>
Function prototype	<code>void dma_channel_enable(dma_channel_type* dmax_channel, confirm_state new_state);</code>
Function description	Enable the selected DMA channel
Input parameter 1	<code>dmax_channel</code> : DMAx_CHANNELy defines a DMA channel number, x=1 or 2, y=1...7
Input parameter 2	<code>new_state</code> : Enable or disable the selected DMA channel
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### **new\_state**

Enable or disable DMA channels

FALSE: Disable

TRUE: Enable

### **Example:**

```
/* enable dma channel */
dma_channel_enable(DMA2_CHANNEL1, TRUE);
```

## 5.8.8 dma\_flag\_get function

The table below describes the function `dma_flag_get`.

**Table 157. `dma_flag_get` function**

Name	Description
Function name	<code>dma_flag_get</code>
Function prototype	<code>flag_status dma_flag_get(uint32_t dmax_flag);</code>
Function description	Get the flag of the selected DMA channel
Input parameter 1	<code>dmax_flag</code> : select the desired flag
Output parameter	NA
Return value	<code>flag_status</code> : indicates whether the desired flag is set or not
Required preconditions	NA
Called functions	NA

### **dmax\_flag**

`dmax_flag` is used for flag section, including:

DMA1_GL1_FLAG:	DMA1 channel 1 global flag
DMA1_FDT1_FLAG:	DMA1 channel 1 transfer complete flag
DMA1_HDT1_FLAG:	DMA1 channel 1 half transfer complete flag
DMA1_DTERR1_FLAG:	DMA1 channel 1 transfer error flag
DMA1_GL2_FLAG:	DMA1 channel 2 global flag
DMA1_FDT2_FLAG:	DMA1 channel 2 transfer complete flag

DMA1_HDT2_FLAG:	DMA1 channel 2 half transfer complete flag
DMA1_DTERR2_FLAG:	DMA1 channel 2 transfer error flag
DMA1_GL3_FLAG:	DMA1 channel 3 global flag
DMA1_FDT3_FLAG:	DMA1 channel 3 transfer complete flag
DMA1_HDT3_FLAG:	DMA1 channel 3 half transfer complete flag
DMA1_DTERR3_FLAG:	DMA1 channel 3 transfer error flag
DMA1_GL4_FLAG:	DMA1 channel 4 global flag
DMA1_FDT4_FLAG:	DMA1 channel 4 transfer complete flag
DMA1_HDT4_FLAG:	DMA1 channel 4 half transfer complete flag
DMA1_DTERR4_FLAG:	DMA1 channel 4 transfer error flag
DMA1_GL5_FLAG:	DMA1 channel 5 global flag
DMA1_FDT5_FLAG:	DMA1 channel 5 transfer complete flag
DMA1_HDT5_FLAG:	DMA1 channel 5 half transfer complete flag
DMA1_DTERR5_FLAG:	DMA1 channel 5 transfer error flag
DMA1_GL6_FLAG:	DMA1 channel 6 global flag
DMA1_FDT6_FLAG:	DMA1 channel 6 transfer complete flag
DMA1_HDT6_FLAG:	DMA1 channel 6 half transfer complete flag
DMA1_DTERR6_FLAG:	DMA1 channel 6 transfer error flag
DMA1_GL7_FLAG:	DMA1 channel 7 global flag
DMA1_FDT7_FLAG:	DMA1 channel 7 transfer complete flag
DMA1_HDT7_FLAG:	DMA1 channel 7 half transfer complete flag
DMA1_DTERR7_FLAG:	DMA1 channel 7 transfer error flag
DMA2_GL1_FLAG:	DMA2 channel 1 global flag
DMA2_FDT1_FLAG:	DMA2 channel 1 transfer complete flag
DMA2_HDT1_FLAG:	DMA2 channel 1 half transfer complete flag
DMA2_DTERR1_FLAG:	DMA2 channel 1 transfer error flag
DMA2_GL2_FLAG:	DMA2 channel 2 global flag
DMA2_FDT2_FLAG:	DMA2 channel 2 transfer complete flag
DMA2_HDT2_FLAG:	DMA2 channel 2 half transfer complete flag
DMA2_DTERR2_FLAG:	DMA2 channel 2 transfer error flag
DMA2_GL3_FLAG:	DMA2 channel 3 global flag
DMA2_FDT3_FLAG:	DMA2 channel 3 transfer complete flag
DMA2_HDT3_FLAG:	DMA2 channel 3 half transfer complete flag
DMA2_DTERR3_FLAG:	DMA2 channel 3 transfer error flag
DMA2_GL4_FLAG:	DMA2 channel 4 global flag
DMA2_FDT4_FLAG:	DMA2 channel 4 transfer complete flag
DMA2_HDT4_FLAG:	DMA2 channel 4 half transfer complete flag
DMA2_DTERR4_FLAG:	DMA2 channel 4 transfer error flag
DMA2_GL5_FLAG:	DMA2 channel 5 global flag
DMA2_FDT5_FLAG:	DMA2 channel 5 transfer complete flag
DMA2_HDT5_FLAG:	DMA2 channel 5 half transfer complete flag
DMA2_DTERR5_FLAG:	DMA2 channel 5 transfer error flag

**flag\_status**

RESET: Flag is reset

SET: Flag is set

**Example:**

```
if(dma_flag_get(DMA1_FDT1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
}
```

## 5.8.9 dma\_flag\_clear function

The table below describes the function `dma_flag_clear`

**Table 158. `dma_flag_clear` function**

Name	Description
Function name	<code>dma_flag_clear</code>
Function prototype	<code>void dma_flag_clear(uint32_t dmax_flag);</code>
Function description	Clear the selected flag
Input parameter 1	<i>dmax_flag</i> : a flag that needs to be cleared
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### **dmax\_flag**

`dmax_flag` is used to select the desired flag, including:

DMA1_GL1_FLAG:	DMA1 channel 1 global flag
DMA1_FDT1_FLAG:	DMA1 channel 1 transfer complete flag
DMA1_HDT1_FLAG:	DMA1 channel 1 half transfer complete flag
DMA1_DTERR1_FLAG:	DMA1 channel 1 transfer error flag
DMA1_GL2_FLAG:	DMA1 channel 2 global flag
DMA1_FDT2_FLAG:	DMA1 channel 2 transfer complete flag
DMA1_HDT2_FLAG:	DMA1 channel 2 half transfer complete flag
DMA1_DTERR2_FLAG:	DMA1 channel 2 transfer error flag
DMA1_GL3_FLAG:	DMA1 channel 3 global flag
DMA1_FDT3_FLAG:	DMA1 channel 3 transfer complete flag
DMA1_HDT3_FLAG:	DMA1 channel 3 half transfer complete flag
DMA1_DTERR3_FLAG:	DMA1 channel 3 transfer error flag
DMA1_GL4_FLAG:	DMA1 channel 4 global flag
DMA1_FDT4_FLAG:	DMA1 channel 4 transfer complete flag
DMA1_HDT4_FLAG:	DMA1 channel 4 half transfer complete flag
DMA1_DTERR4_FLAG:	DMA1 channel 4 transfer error flag
DMA1_GL5_FLAG:	DMA1 channel 5 global flag
DMA1_FDT5_FLAG:	DMA1 channel 5 transfer complete flag
DMA1_HDT5_FLAG:	DMA1 channel 5 half transfer complete flag
DMA1_DTERR5_FLAG:	DMA1 channel 5 transfer error flag
DMA1_GL6_FLAG:	DMA1 channel 6 global flag
DMA1_FDT6_FLAG:	DMA1 channel 6 transfer complete flag
DMA1_HDT6_FLAG:	DMA1 channel 6 half transfer complete flag
DMA1_DTERR6_FLAG:	DMA1 channel 6 transfer error flag
DMA1_GL7_FLAG:	DMA1 channel 7 global flag
DMA1_FDT7_FLAG:	DMA1 channel 7 transfer complete flag
DMA1_HDT7_FLAG:	DMA1 channel 7 half transfer complete flag
DMA1_DTERR7_FLAG:	DMA1 channel 7 transfer error flag
DMA2_GL1_FLAG:	DMA2 channel 1 global flag
DMA2_FDT1_FLAG:	DMA2 channel 1 transfer complete flag
DMA2_HDT1_FLAG:	DMA2 channel 1 half transfer complete flag

DMA2_DTERR1_FLAG:	DMA2 channel 1 transfer error flag
DMA2_GL2_FLAG:	DMA2 channel 2 global flag
DMA2_FDT2_FLAG:	DMA2 channel 2 transfer complete flag
DMA2_HDT2_FLAG:	DMA2 channel 2 half transfer complete flag
DMA2_DTERR2_FLAG:	DMA2 channel 2 transfer error flag
DMA2_GL3_FLAG:	DMA2 channel 3 global flag
DMA2_FDT3_FLAG:	DMA2 channel 3 transfer complete flag
DMA2_HDT3_FLAG:	DMA2 channel 3 half transfer complete flag
DMA2_DTERR3_FLAG:	DMA2 channel 3 transfer error flag
DMA2_GL4_FLAG:	DMA2 channel 4 global flag
DMA2_FDT4_FLAG:	DMA2 channel 4 transfer complete flag
DMA2_HDT4_FLAG:	DMA2 channel 4 half transfer complete flag
DMA2_DTERR4_FLAG:	DMA2 channel 4 transfer error flag
DMA2_GL5_FLAG:	DMA2 channel 5 global flag
DMA2_FDT5_FLAG:	DMA2 channel 5 transfer complete flag
DMA2_HDT5_FLAG:	DMA2 channel 5 half transfer complete flag
DMA2_DTERR5_FLAG:	DMA2 channel 5 transfer error flag

```
if(dma_flag_get(DMA2_FDT1_FLAG) != RESET)
{
    /* turn led2/led3/led4 on */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
    dma_flag_clear(DMA2_FDT1_FLAG);
}
```

## 5.9 External interrupt/event controller (EXINT)

The EXINT register structure exint\_type is defined in the “at32f403\_exint.h”:

```
/*
 * @brief type define exint register all
 */
typedef struct
{
    ...
} exint_type;
```

The table below gives a list of the EXINT registers.

**Table 159. Summary of EXINT registers**

Register	Description
inten	Interrupt enable register
evten	Event enable register
polcfg1	Polarity configuration register 1
polcfg2	Polarity configuration register 2
swtrg	Software trigger register
intsts	Interrupt status register

The table below gives a list of EXINT library functions.

**Table 160. Summary of EXINT library functions**

Function name	Description
exint_reset	Reset all EXINT registers to their reset values.
exint_default_para_init	Configure the EXINT initial structure with the initial value
exint_init	Initialize EXINT
exint_flag_clear	Clear the selected EXINT interrupt flag
exint_flag_get	Read the selected EXINT interrupt flag
exint_software_interrupt_event_generate	Software interrupt event generation
exint_interrupt_enable	Enable the selected EXINT interrupt
exint_event_enable	Enable the selected EXINT event

## 5.9.1 exint\_reset function

The table below describes the function exint\_reset.

**Table 161. exint\_reset function**

Name	Description
Function name	exint_reset
Function prototype	void exint_reset(void);
Function description	Reset all EXINT registers to their reset values.
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

**Example:**

```
exint_reset();
```

## 5.9.2 exint\_default\_para\_init function

The table below describes the function exint\_default\_para\_init

**Table 162. exint\_default\_para\_init function**

Name	Description
Function name	exint_default_para_init
Function prototype	void exint_default_para_init(exint_init_type *exint_struct);
Function description	Configure the EXINT initial structure with the initiali value
Input parameter 1	exint_struct: <a href="#">exint_init_type</a> pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of exint_init_type before starting.
Called functions	NA

**Example:**

```
exint_init_type exint_init_struct;  
exint_default_para_init(&exint_init_struct);
```

### 5.9.3 exint\_init function

The table below describes the function exint\_init

**Table 163. exint\_init function**

Name	Description
Function name	exint_init
Function prototype	void exint_init(exint_init_type *exint_struct);
Function description	Initialize EXINT
Input parameter 1	<i>exint_init_type</i> : exint_init_struct pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of exint_init_type before starting.
Called functions	NA

exint\_init\_type is defined in the at32f403\_exint.h:

```
typedef struct
{
    exint_line_mode_type          line_mode;
    uint32_t                      line_select;
    exint_polarity_config_type   line_polarity;
    confirm_state                 line_enable;
} exint_init_type;
```

#### line\_mode

Select event mode or interrupt mode

EXINT_LINE_INTERRUPT:	Interrupt mode
EXINT_LINE_EVENT:	Event mode

#### line\_select

Line selection.

EXINT_LINE_NONE:	No line
EXINT_LINE_0:	line0
EXINT_LINE_1:	line1
...	
EXINT_LINE_18:	line18
EXINT_LINE_19:	line19

#### line\_polarity

Trigger edge selection.

EXINT_TRIGGER_RISING_EDGE:	Rising edge
EXINT_TRIGGER_FALLING_EDGE:	Falling edge
EXINT_TRIGGER_BOTH_EDGE:	Rising/falling edge

#### line\_enable

Enable/disable line

FALSE: Disable line

TRUE: Enable line

#### Example:

```
exint_init_type exint_init_struct;
exint_default_para_init(&exint_init_struct);
```

```

exint_init_struct.line_enable = TRUE;
exint_init_struct.line_mode = EXINT_LINE_INTERRUPT;
exint_init_struct.line_select = EXINT_LINE_0;
exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
exint_init(&exint_init_struct);

```

## 5.9.4 exint\_flag\_clear function

The table below describes the function exint\_flag\_clear

**Table 164. exint\_flag\_clear function**

Name	Description
Function name	exint_flag_clear
Function prototype	void exint_flag_clear(uint32_t exint_line);
Function description	Clear the selected EXINT interrupt flag
Input parameter	exint_line: line selection Refer to the " <a href="#">line_select</a> " for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
exint_flag_clear(EXINT_LINE_0);
```

## 5.9.5 exint\_flag\_get function

The table below describes the function exint\_flag\_get.

**Table 165. exint\_flag\_get function**

Name	Description
Function name	exint_flag_get
Function prototype	flag_status exint_flag_get(uint32_t exint_line);
Function description	Get the selected EXINT interrupt flag
Input parameter	exint_line: line selection Refer to the " <a href="#">line_select</a> " for details.
Output parameter	NA
Return value	flag_status: indicates the status of the selected flag This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

**Example:**

```

flag_status status = RESET;
status = exint_flag_get(EXINT_LINE_0);

```

## 5.9.6 exint\_interrupt\_flag\_get function

The table below describes the function exint\_interrupt\_flag\_get.

**Table 166. exint\_interrupt\_flag\_get function**

Name	Description
Function name	exint_interrupt_flag_get
Function prototype	flag_status exint_interrupt_flag_get(uint32_t exint_line)
Function description	Get the selected EXINT interrupt flag
Input parameter	exint_line: line selection Refer to the <a href="#">line_select</a> for details.
Output parameter	NA
Return value	flag_status: indicates the status of the selected flag This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

**Example**

```
flag_status status = RESET;
status = exint_interrupt_flag_get (EXINT_LINE_0);
```

## 5.9.7 exint\_software\_interrupt\_event\_generate function

The table below describes the function exint\_software\_interrupt\_event\_generate.

**Table 167. exint\_software\_interrupt\_event\_generate function**

Name	Description
Function name	exint_software_interrupt_event_generate
Function prototype	void exint_software_interrupt_event_generate(uint32_t exint_line);
Function description	Generate software interrupt event
Input parameter	exint_line: line selection Refer to the <a href="#">“line_select”</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
exint_software_interrupt_event_generate (EXINT_LINE_0);
```

## 5.9.8 exint\_interrupt\_enable function

The table below describes the function exint\_interrupt\_enable.

**Table 168. exint\_interrupt\_enable function**

Name	Description
Function name	exint_interrupt_enable
Function prototype	void exint_interrupt_enable(uint32_t exint_line, confirm_state new_state);
Function description	Enable the selected EXINT interrupt
Input parameter 1	exint_line: line selection Refer to the " <a href="#">line_select</a> " for details.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
exint_interrupt_enable (EXINT_LINE_0);
```

## 5.9.9 exint\_event\_enable function

The table below describes the function exint\_event\_enable.

**Table 169. exint\_event\_enable function**

Name	Description
Function name	exint_event_enable
Function prototype	void exint_event_enable(uint32_t exint_line, confirm_state new_state);
Function description	Enable the selected EXINT event
Input parameter 1	exint_line: line selection Refer to the " <a href="#">line_select</a> " for details
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
exint_event_enable (EXINT_LINE_0);
```

## 5.10 Flash memory controller (FLASH)

The FLASH register structure flash\_type is defined in the “at32f403\_flash.h”:

```
/**  
 * @brief type define flash register all  
 */  
  
typedef struct  
{  
    ...  
} flash_type;
```

The table below gives a list of the FLASH registers.

Table 170. Summary of FLASH registers

Register	Description
flash_psr	Flash performance select register
flash_unlock	Flash unlock register
flash_usd_unlock	Flash user system data unlock register
flash_sts	Flash status register
flash_ctrl	Flash control register
flash_addr	Flash address register
flash_usd	User system data register
flash_epps	Erase/program protection status register
flash_unlock2	Flash unlock register2
flash_sts2	Flash status register2
flash_ctrl2	Flash control register2
flash_addr2	Flash address register2
flash_unlock3	Flash unlock register3
flash_select	Flash select register
flash_sts3	Flash status register3
flash_ctrl3	Flash control register3
flash_addr3	Flash address register3
flash_da	Flash decription address register

The table below gives a list of FLASH library functions.

**Table 171. Summary of FLASH library functions**

Function name	Description
flash_flag_get	Get flag status
flash_flag_clear	Clear flag
flash_operation_status_get	Get operation status (Flash memory bank 1)
flash_operation_wait_for	Wait for operation complete (Flash memory bank 1)
flash_bank1_operation_status_get	Get Flash bank1 operation status
flash_bank2_operation_status_get	Get Flash bank2 operation status
flash_spim_operation_status_get	Get SPIM operation status
flash_operation_wait_for	Wait for operation complete (Flash bank 1)
flash_bank1_operation_wait_for	Wait for Flash bank 1 operation complete
flash_bank2_operation_wait_for	Wait for Flash bank 2 operation complete
flash_spim_operation_wait_for	Wait for SPIM operation complete
flash_unlock	Unlock Flash (Flash memory bank 1 and 2)
flash_bank1_unlock	Unlock Flash bank 1
flash_bank2_unlock	Unlock Flash bank 2
flash_spim_unlock	Unlock SPIM
flash_lock	Lock Flash (Flash memory bank 1 and 2)
flash_bank1_lock	Lock 1 Flash bank 1
flash_bank2_lock	Lock Flash bank 2
flash_spim_lock	Lock SPIM
flash_sector_erase	Erase Flash sector
flash_internal_all_erase	Erase internal Flash
flash_bank1_erase	Erase Flash bank 1
flash_bank2_erase	Erase Flash bank 2
flash_spim_all_erase	Erase SPIM
flash_user_system_data_erase	Erase user system data
flash_word_program	Flash word programming
flash_halfword_program	Flash half-word programming
flash_byte_program	Flash byte programming
flash_user_system_data_program	User system data programming
flash_user_system_data_erase	Erase user system data area
flash_word_program	Flash word programming
flash_halfword_program	Flash half-word programming
flash_byte_program	Flash byte programming
flash_epp_set	Erase/programming protection configuration
flash_epp_status_get	Get erase/programming protection status
flash_fap_enable	Flash access protection enable
flash_fap_status_get	Get Flash access protection status
flash_ssb_set	System configuration byte configuration
flash_ssb_status_get	Get system configuration byte configuration status
flash_interrupt_enable	Flash interrupt configuration
flash_spim_model_select	SPIM model selection

Function name	Description
flash_spim_encryption_range_set	SPIM encryption range configuration

### 5.10.1 flash\_flag\_get function

The table below describes the function flash\_flag\_get.

Table 172. flash\_flag\_get function

Name	Description
Function name	flash_flag_get
Function prototype	flag_status flash_flag_get(uint32_t flash_flag);
Function description	Get flag status
Input parameter	flash_flag: Flag selection
Output parameter	NA
Return value	flag_status: indicates the flag status Return RESET or SET.
Required preconditions	NA
Called functions	NA

#### flash\_flag

Flag selection.

FLASH_OBF_FLAG:	Flash operation busy (bank 1)
FLASH_ODF_FLAG:	Flash operation complete (bank 1)
FLASH_PGMERR_FLAG:	Flash programming error (bank 1)
FLASH_EPPERR_FLAG:	Flash erase error (bank 1)
FLASH_BANK1_OBF_FLAG:	Flash bank 1 operation busy
FLASH_BANK1_ODF_FLAG:	Flash bank 1 operation complete
FLASH_BANK1_PGMERR_FLAG:	Flash bank 1 programming error
FLASH_BANK1_EPPERR_FLAG:	Flash bank 1 erase error
FLASH_BANK2_OBF_FLAG:	Flash bank 2 operation busy
FLASH_BANK2_ODF_FLAG:	Flash bank 2 operation complete
FLASH_BANK2_PGMERR_FLAG:	Flash bank 2 programming error
FLASH_BANK2_EPPERR_FLAG:	Flash bank 2 erase error
FLASH_SPIM_OBF_FLAG:	SPIM operation busy
FLASH_SPIM_ODF_FLAG:	SPIM operation complete
FLASH_SPIM_PGMERR_FLAG:	SPIM programming error
FLASH_SPIM_EPPERR_FLAG:	SPIM erase error
FLASH_USDERR_FLAG:	User system data area error

#### Example:

```
flag_status status;
status = flash_flag_get(FLASH_ODF_FLAG);
```

### 5.10.2 flash\_flag\_clear function

The table below describes the function flash\_flag\_clear.

Table 173. flash\_flag\_clear function

Name	Description
Function name	flash_flag_clear
Function prototype	void flash_flag_clear(uint32_t flash_flag);
Function description	Clear flag
Input parameter	flash_flag: flag selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**flash\_flag**

Flag selection

FLASH_ODF_FLAG:	Flash operation complete (bank 1)
FLASH_PRGMERR_FLAG:	Flash programming error (bank 1)
FLASH_EPPERR_FLAG:	Flash erase error (bank 1)
FLASH_BANK1_ODF_FLAG:	Flash bank 1 operation complete
FLASH_BANK1_PRGMERR_FLAG:	Flash bank 1 programming error
FLASH_BANK1_EPPERR_FLAG:	Flash bank 1 erase error
FLASH_BANK2_ODF_FLAG:	Flash bank 2 operation complete
FLASH_BANK2_PRGMERR_FLAG:	Flash bank 2 programming error
FLASH_BANK2_EPPERR_FLAG:	Flash bank 2 erase error
FLASH_SPIM_ODF_FLAG:	SPIIM operation complete
FLASH_SPIM_PRGMERR_FLAG:	SPIIM programming error
FLASH_SPIM_EPPERR_FLAG:	SPIIM erase error

**Example:**

```
flash_flag_clear(FLASH_ODF_FLAG);
```

### 5.10.3 flash\_operation\_status\_get function

The table below describes the function flash\_operation\_status\_get.

**Table 174. flash\_operation\_status\_get function**

Name	Description
Function name	flash_operation_status_get
Function prototype	flash_status_type flash_operation_status_get(void);
Function description	Get operation status
Input parameter	NA
Output parameter	NA
Return value	Refer to the <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

#### flash\_status\_type

FLASH_OPERATE_BUSY	Operate busy
FLASH_PROGRAM_ERROR	Programming error
FLASH_EPP_ERROR	Erase/program protection error
FLASH_OPERATE_DONE	Operation complete
FLASH_OPERATE_TIMEOUT	Operation timeout

#### Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
/* check for the flash status */
status = flash_operation_status_get();
```

### 5.10.4 flash\_bank1\_operation\_status\_get function

The table below describes the function flash\_bank1\_operation\_status\_get

**Table 175. flash\_bank1\_operation\_status\_get function**

Name	Description
Function name	flash_bank1_operation_status_get
Function prototype	flash_status_type flash_bank1_operation_status_get (void);
Function description	Get the status of Flash bank 1
Input parameter	NA
Output parameter	NA
Return value	Return Flash bank 1 operation status, refer to the <a href="#">flash_status_type</a> for details
Required preconditions	NA
Called functions	NA

#### Example:

```
flash_status_type status = FLASH_OPERATE_DONE;
/* check for the flash status */
status = flash_bank1_operation_status_get();
```

## 5.10.5 flash\_bank2\_operation\_status\_get function

The table below describes the function flash\_bank2\_operation\_status\_get.

**Table 176. flash\_bank2\_operation\_status\_get function**

Name	Description
Function name	flash_bank2_operation_status_get
Function prototype	flash_status_type flash_bank2_operation_status_get (void);
Function description	Get the status of Flash bank 2
Input parameter	NA
Output parameter	NA
Return value	Return Flash bank 2 operation status, refer to the <a href="#">flash_status_type</a> for details
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
/* check for the flash status */
status = flash_bank2_operation_status_get();
```

## 5.10.6 flash\_spim\_operation\_status\_get function

The table below describes the function flash\_spim\_operation\_status\_get.

**Table 177. flash\_spim\_operation\_status\_get function**

Name	Description
Function name	flash_spim_operation_status_get
Function prototype	flash_status_type flash_spim_operation_status_get (void);
Function description	Get the status of SPIM (external Flash)
Input parameter	NA
Output parameter	NA
Return value	Return Flash bank 2 operation status, refer to the <a href="#">flash_status_type</a> for details
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
/* check for the flash status */
status = flash_spim_operation_status_get();
```

## 5.10.7 flash\_operation\_wait\_for function

The table below describes the function flash\_operation\_wait\_for.

**Table 178. flash\_operation\_wait\_for function**

Name	Description
Function name	flash_operation_wait_for
Function prototype	flash_status_type flash_operation_wait_for(uint32_t time_out);
Function description	Wait for Flash operation
Input parameter	time_out: wait timeout The timeout value is defined in the flash.h file, refer to <a href="#">flash_time_out</a> for details.
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

### flash\_time\_out

ERASE_TIMEOUT	Erase timeout
PROGRAMMING_TIMEOUT	Programming timeout
SPIM_ERASE_TIMEOUT:	SPIM erase timeout
SPIM_PROGRAMMING_TIMEOUT:	SPIM programming timeout
OPERATION_TIMEOUT	General operation timeout

### Example:

```
/* wait for operation to be completed */
status = flash_operation_wait_for(PROGRAMMING_TIMEOUT);
```

## 5.10.8 flash\_bank1\_operation\_wait\_for function

The table below describes the function flash\_bank1\_operation\_wait\_for.

**Table 179. flash\_bank1\_operation\_wait\_for function**

Name	Description
Function name	flash_bank1_operation_wait_for
Function prototype	flash_status_type flash_bank1_operation_wait_for(uint32_t time_out);
Function description	Wait for Flash bank1 operation
Input parameter	time_out: wait timeout The timeout value is defined in the flash.h file, refer to <a href="#">flash_time_out</a> for details.
Output parameter	NA
Return value	Return Flash bank 2 operation status, refer to the <a href="#">flash_status_type</a> for details
Required preconditions	NA
Called functions	NA

**Example:**

```
/* wait for operation to be completed */
status = flash_bank1_operation_wait_for(PROGRAMMING_TIMEOUT);
```

## 5.10.9 flash\_bank2\_operation\_wait\_for function

The table below describes the function flash\_bank2\_operation\_wait\_for.

**Table 180. flash\_bank2\_operation\_wait\_for function**

Name	Description
Function name	flash_bank2_operation_wait_for
Function prototype	flash_status_type flash_bank2_operation_wait_for(uint32_t time_out);
Function description	Wait for Flash bank2 operation
Input parameter	time_out: wait timeout The timeout value is defined in the flash.h file, refer to <a href="#">flash_time_out</a> for details.
Output parameter	NA
Return value	Return Flash bank 2 operation status, refer to the <a href="#">flash_status_type</a> for details
Required preconditions	NA
Called functions	NA

**Example:**

```
/* wait for operation to be completed */
status = flash_bank2_operation_wait_for(PROGRAMMING_TIMEOUT);
```

### 5.10.10 flash\_spim\_operation\_wait\_for function

The table below describes the function flash\_spim\_operation\_wait\_for.

**Table 181. flash\_spim\_operation\_wait\_for function**

Name	Description
Function name	flash_spim_operation_wait_for
Function prototype	flash_status_type flash_spim_operation_wait_for(uint32_t time_out);
Function description	Wait for SPIM (external Flash) operation
Input parameter	time_out: wait timeout The timeout value is defined in the flash.h file, refer to <a href="#">flash_time_out</a> for details.
Output parameter	NA
Return value	Return Flash bank 2 operation status, refer to the <a href="#">flash_status_type</a> for details
Required preconditions	NA
Called functions	NA

**Example:**

```
/* wait for operation to be completed */
status = flash_spim_operation_wait_for(PROGRAMMING_TIMEOUT);
```

### 5.10.11 flash\_unlock function

The table below describes the function flash\_unlock.

**Table 182. flash\_unlock function**

Name	Description
Function name	flash_unlock
Function prototype	void flash_unlock(void);
Function description	Unlock Flash memory controller
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_unlock();
```

### 5.10.12 flash\_bank1\_unlock function

The table below describes the function flash\_bank1\_unlock.

**Table 183. flash\_bank1\_unlock function**

Name	Description
Function name	flash_bank1_unlock
Function prototype	void flash_bank1_unlock(void);
Function description	Unlock Flash bank1 control register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_bank1_unlock();
```

### 5.10.13 flash\_bank2\_unlock function

The table below describes the function flash\_bank2\_unlock.

**Table 184. flash\_bank2\_unlock function**

Name	Description
Function name	flash_bank2_unlock
Function prototype	void flash_bank2_unlock(void);
Function description	Unlock Flash bank2 control register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_bank2_unlock();
```

### 5.10.14 flash\_spim\_unlock function

The table below describes the function flash\_spim\_unlock.

**Table 185. flash\_spim\_unlock function**

Name	Description
Function name	flash_spim_unlock
Function prototype	void flash_spim_unlock(void);
Function description	Unlock SPIM (external Flash) control register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_spim_unlock();
```

### 5.10.15 flash\_lock function

The table below describes the function flash\_lock.

**Table 186. flash\_lock function**

Name	Description
Function name	flash_lock
Function prototype	void flash_lock(void);
Function description	Lock Flash memory controller
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_lock();
```

### 5.10.16 flash\_bank1\_lock function

The table below describes the function flash\_bank1\_lock.

**Table 187. flash\_bank1\_lock function**

Name	Description
Function name	flash_bank1_lock
Function prototype	void flash_bank1_lock(void);
Function description	Lock Flash bank1 controll register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_bank1_lock();
```

### 5.10.17 flash\_bank2\_lock function

The table below describes the function flash\_bank2\_lock.

**Table 188. flash\_bank2\_lock function**

Name	Description
Function name	flash_bank2_lock
Function prototype	void flash_bank2_lock(void);
Function description	Lock Flash bank2 controll register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_bank2_lock();
```

### 5.10.18 flash\_spim\_lock function

The table below describes the function flash\_spim\_lock.

**Table 189. flash\_spim\_lock function**

Name	Description
Function name	flash_spim_lock
Function prototype	void flash_spim_lock(void);
Function description	Lock SPIM (external Flash) control register
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_spim_lock();
```

### 5.10.19 flash\_sector\_erase function

The table below describes the function flash\_sector\_erase.

**Table 190. flash\_sector\_erase function**

Name	Description
Function name	flash_sector_erase
Function prototype	flash_status_type flash_sector_erase(uint32_t sector_address);
Function description	Erase data in the selected Flash sector address
Input parameter	sector_address: select the Flash sector address to be erased, usually Flash sector start address
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;  
flash_unlock();  
status = flash_sector_erase(0x08001000);
```

### 5.10.20 flash\_internal\_all\_erase function

The table below describes the function flash\_internal\_all\_erase.

**Table 191. flash\_internal\_all\_erase function**

Name	Description
Function name	flash_internal_all_erase
Function prototype	flash_status_type flash_internal_all_erase(void);
Function description	Erase internal Flash data
Input parameter	NA
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;  
flash_unlock();  
status = flash_internal_all_erase();
```

## 5.10.21 flash\_bank1\_erase function

The table below describes the function flash\_bank1\_erase.

**Table 192. flash\_bank1\_erase function**

Name	Description
Function name	flash_bank1_erase
Function prototype	flash_status_type flash_bank1_erase(void);
Function description	Erase internal Flash bank1 data
Input parameter	NA
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;  
flash_bank1_unlock();  
status = flash_bank1_erase();
```

## 5.10.22 flash\_bank2\_erase function

The table below describes the function flash\_bank2\_erase.

**Table 193. flash\_bank2\_erase function**

Name	Description
Function name	flash_bank2_erase
Function prototype	flash_status_type flash_bank2_erase(void);
Function description	Erase internal Flash bank2 data
Input parameter	NA
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;  
flash_bank2_unlock();  
status = flash_bank2_erase();
```

### 5.10.23 flash\_spim\_all\_erase function

The table below describes the function flash\_spim\_all\_erase.

**Table 194. flash\_spim\_all\_erase function**

Name	Description
Function name	flash_spim_all_erase
Function prototype	flash_status_type flash_spim_all_erase(void);
Function description	Erase external Flash (SPIM) data
Input parameter	NA
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;  
flash_spim_unlock();  
status = flash_spim_all_erase();
```

### 5.10.24 flash\_user\_system\_data\_erase function

The table below describes the function flash\_user\_system\_data\_erase.

**Table 195. flash\_user\_system\_data\_erase function**

Name	Description
Function name	flash_user_system_data_erase
Function prototype	flash_status_type flash_user_system_data_erase(void);
Function description	Erase user system data
Input parameter	NA
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

*Note: As this function remains in FAP state, it only erases data except FAP in the user system data area.*

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;  
flash_unlock();  
status = flash_user_system_data_erase();
```

## 5.10.25 flash\_word\_program function

The table below describes the function flash\_word\_program.

**Table 196. flash\_word\_program function**

Name	Description
Function name	flash_word_program
Function prototype	flash_status_type flash_word_program(uint32_t address, uint32_t data);
Function description	Program a word data to a given address
Input parameter 1	Address: programming address, word-aligned
Input parameter 2	Data: programming data
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	The programming operation can be allowed only when data in the address are all 0xFF
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
uint32_t i;
flash_unlock();
status = flash_sector_erase(0x08001000);
if(status = FLASH_OPERATE_DONE)
{
    /* program 256 words */
    for(I = 0; I < 256; i++)
    {
        status = flash_word_program(0x08001000 + i*4, i);
    }
}
```

## 5.10.26 flash\_halfword\_program function

The table below describes the function flash\_halfword\_program.

**Table 197. flash\_halfword\_program function**

Name	Description
Function name	flash_halfword_program
Function prototype	flash_status_type flash_halfword_program(uint32_t address, uint16_t data);
Function description	Program a half-word data to a given address
Input parameter 1	Address: programming address, half-word-aligned
Input parameter 2	Data: programming data
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	The programming operation can be allowed only when data in the address are all 0xFF
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
```

```
uint32_t i;
flash_unlock();
status = flash_sector_erase(0x08001000);
if(status == FLASH_OPERATE_DONE)
{
    /* program 256 halfwords */
    for(i = 0; i < 256; i++)
    {
        status = flash_halfword_program(0x08001000 + i*2, (uint16_t)i);
    }
}
```

### 5.10.27 flash\_byte\_program function

The table below describes the function flash\_byte\_program.

**Table 198. flash\_byte\_program function**

Name	Description
Function name	flash_byte_program
Function prototype	flash_status_type flash_byte_program(uint32_t address, uint8_t data);
Function description	Program a byte data to a given address
Input parameter 1	Address: programming address
Input parameter 2	Data: programming data
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	The programming operation can be allowed only when data in the address are all 0xFF
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
uint32_t i;
flash_unlock();
status = flash_sector_erase(0x08001000);
if(status == FLASH_OPERATE_DONE)
{
    /* program 256 bytes */
    for(i = 0; i < 256; i++)
    {
        status = flash_byte_program(0x08001000 + i*2, (uint8_t)i);
    }
}
```

## 5.10.28 flash\_user\_system\_data\_program function

The table below describes the function flash\_user\_system\_data\_program.

**Table 199. flash\_user\_system\_data\_program function**

Name	Description
Function name	flash_user_system_data_program
Function prototype	flash_status_type flash_user_system_data_program (uint32_t address, uint8_t data);
Function description	Program a byte data to a given address in the user system data area
Input parameter 1	Address: programmed address
Input parameter 2	Data: programmed data
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	The programming operation can be allowed only when data and its inverse data in the user system data area are all 0xFF
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_user_system_data_erase();
if(status = FLASH_OPERATE_DONE)
{
    /* program user system data */
    status = flash_user_system_data_program(0x1FFFF804, 0x55);
}
```

## 5.10.29 flash\_epp\_set function

The table below describes the function flash\_epp\_set.

**Table 200. flash\_epp\_set function**

Name	Description
Function name	flash_epp_set
Function prototype	flash_status_type flash_epp_set(uint32_t *sector_bits);
Function description	Enable erase programming protection
Input parameter	*sector_bits: Erase programming protection sector address pointer. Each bit protects 4KB sectors, and the last bit guards the remaining sectors. When this bit is set to 1, it indicates that the corresponding sector is protected.
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
uint32_t epp_val;
```

```
flash_unlock();
status = flash_user_system_data_erase();
if(status == FLASH_OPERATE_DONE)
{
    epp_val = 0x00000001;
    /* program epp */
    status = flash_epp_set(&epp_val);
}
```

### 5.10.30 flash\_epp\_status\_get function

The table below describes the function flash\_epp\_status\_get.

Table 201. flash\_epp\_status\_get function

Name	Description
Function name	flash_epp_status_get
Function prototype	void flash_epp_status_get(uint32_t *sector_bits);
Function description	Get the status of erase programming protection
Input parameter	NA
Output parameter	*sector_bits: Erase programming protection sector address pointer. Each bit protects 4KB sectors, and the last bit guards the remaining sectors. When this bit is set to 1, it indicates that the corresponding sector is protected.
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
uint32_t epp_val[2];
/* get epp status */
flash_epp_status_get(epp_val);
```

### 5.10.31 flash\_fap\_enable function

The table below describes the function flash\_fap\_enable.

**Table 202. flash\_fap\_enable function**

Name	Description
Function name	flash_fap_enable
Function prototype	flash_status_type flash_fap_enable(confirm_state new_state);
Function description	Enable Flash access protection
Input parameter	new_state: Flash access protection status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	Refer to <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

*Note: This function will erase the whole user system data area. If there were data programmed in the user system data area before calling this function, they have to be re-programmed after calling this function.*

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_fap_enable(TRUE);
```

### 5.10.32 flash\_fap\_status\_get function

The table below describes the function flash\_fap\_status\_get.

**Table 203. flash\_fap\_status\_get function**

Name	Description
Function name	flash_fap_status_get
Function prototype	flag_status flash_fap_status_get(void);
Function description	Get the status of Flash access protection
Input parameter	NA
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

**Example:**

```
flag_status status;
status = flash_fap_status_get();
```

### 5.10.33 flash\_ssb\_set function

The table below describes the function flash\_ssb\_set.

**Table 204. flash\_ssb\_set function**

Name	Description
Function name	flash_ssb_set
Function prototype	flash_status_type flash_ssb_set(uint8_t usd(ssb));
Function description	Configure system setting bytes
Input parameter	usd(ssb): system setting byte value is a combination of the selected data from all data group, refer to <a href="#">ssb_data_define</a> for details.
Output parameter	NA
Return value	Return operation status, refer to the <a href="#">flash_status_type</a> for details.
Required preconditions	NA
Called functions	NA

#### ssb\_data\_define

type 1:

USD_WDT_ATO_DISABLE:	Watchdog auto-start disabled
USD_WDT_ATO_ENABLE:	Watchdog auto-start enabled

type 2:

USD_DEPSLP_NO_RST:	No reset occurred when entering Deepsleep mode
USD_DEPSLP_RST:	Reset occurred when entering Deepsleep mode

type 3:

USD_STDBY_NO_RST:	No reset occurred when entering Standby mode
USD_STDBY_RST:	Reset occurred when entering Standby mode

type 4:

FLASH_BOOT_FROM_BANK1:	Boot from Flash bank 1
FLASH_BOOT_FROM_BANK2:	Boot from Flash bank 2

#### Example:

```

flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_user_system_data_erase();
if(status == FLASH_OPERATE_DONE)
{
    status = flash_ssb_set(USD_WDT_ATO_DISABLE | USD_DEPSLP_NO_RST | USD_STDBY_RST |
FLASH_BOOT_FROM_BANK1);
}

```

### 5.10.34 flash\_ss\_status\_get function

The table below describes the function flash\_ss\_status\_get.

**Table 205. flash\_ss\_status\_get function**

Name	Description
Function name	flash_ss_status_get
Function prototype	uint8_t flash_ss_status_get(void);
Function description	Get the status of system setting bytes
Input parameter	NA
Output parameter	NA
Return value	Return system setting byte value, refer to <a href="#">ssb_data_define</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
uint8_t ssb_val;
ssb_val = flash_ss_status_get();
```

### 5.10.35 flash\_interrupt\_enable function

The table below describes the function flash\_interrupt\_enable.

**Table 206. flash\_interrupt\_enable function**

Name	Description
Function name	flash_interrupt_enable
Function prototype	void flash_interrupt_enable(uint32_t flash_int, confirm_state new_state);
Function description	Enable Flash interrupts
Input parameter 1	flash_int: Flash interrupt type. Refer to <a href="#">flash_interrupt_type</a> for details.
Input parameter 2	new_state: interrupt status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**flash\_interrupt\_type**

- FLASH\_ERR\_INT: Flash error interrupt
- FLASH\_ODF\_INT: Flash operation complete interrupt
- FLASH\_BANK1\_ERR\_INT: Flash bank 1 error interrupt
- FLASH\_BANK1\_ODF\_INT: Flash bank 1 operation complete interrupt
- FLASH\_BANK2\_ERR\_INT: Flash bank 2 error interrupt
- FLASH\_BANK2\_ODF\_INT: Flash bank 2 operation complete interrupt
- FLASH\_SPIM\_ERR\_INT: SPI-M error interrupt
- FLASH\_SPIM\_ODF\_INT: SPI-M operation complete interrupt

**Example:**

```
flash_interrupt_enable(FLASH_BANK1_ERR_INT | FLASH_BANK1_ODF_INT, TRUE);
```

### 5.10.36 flash\_spim\_model\_select function

The table below describes the function flash\_spim\_model\_select.

**Table 207. flash\_spim\_model\_select function**

Name	Description
Function name	flash_spim_model_select
Function prototype	void flash_spim_model_select(flash_spim_model_type mode);
Function description	Select SPIM mode
Input parameter	Mode: SPIM mode, refer to the <a href="#">flash_spim_mode_type</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**flash\_spim\_mode\_type**

FLASH\_SPIM\_MODEL1: SPIM mode 1

FLASH\_SPIM\_MODEL2: SPIM mode 2

**Example:**

```
flash_spim_model_select(FLASH_SPIM_MODEL1);
```

### 5.10.37 flash\_spim\_encryption\_range\_set function

The table below describes the function flash\_spim\_encryption\_range\_set.

**Table 208. flash\_spim\_encryption\_range\_set function**

Name	Description
Function name	flash_spim_encryption_range_set
Function prototype	void flash_spim_encryption_range_set(uint32_t decode_address);
Function description	Configure SPIM data encryption data
Input parameter	decode_address: address range for encryption, word-aligned, meaning that the data located before this address are stored as ciphertext.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
flash_spim_encryption_range_set(0x08401000);
```

## 5.11 Geneal-purpose and multiplexed I/Os (GPIO/IOMUX)

The GPIO and IOMUX register structure gpio\_type and iomux\_type is defined in the “at32f403\_gpio.h”:

```
/*
 * @brief type define gpio register all
 */
typedef struct
{

} gpio_type;

/*
 * @brief type define iomux register all
 */
typedef struct
{

} iomux_type;
```

The table below gives a list of the GPIO registers.

**Table 209. Summary of GPIO registers**

Register	Description
cfglr	GPIO configuration register low
cfghr	GPIO configuration register high
idt	GPIO input register
odt	GPIO output register
scr	GPIO set/clear register
clr	GPIO bit clear register
wpr	GPIO write protection register

The table below gives a list of IOMUX library functions.

**Table 210. Summary of IOMUX registers**

Register	Description
evtout	Event output control register
remap	IOMUX remap register
exintc1	IOMUX external interrupt configuration register 1
exintc2	IOMUX external interrupt configuration register 2
exintc3	IOMUX external interrupt configuration register 3
exintc4	IOMUX external interrupt configuration register 4
remap2	IOMUX remap register 2

The table below gives a list of GPIO and IOMUX library functions.

Table 211. GPIO and IOMUX library functions

Function name	Description
gpio_reset	GPIO is reset by CRM reset register
gpio_iomux_reset	IOMUX is reset by CRM reset register
gpio_init	Initialize GPIO peripherals
gpio_default_para_init	Initialize GPIO default parameters
gpio_input_data_bit_read	Read GPIO input data bit
gpio_input_data_read	Read GPIO input data
gpio_output_data_bit_read	Read GPIO output data bit
gpio_output_data_read	Read GPIO output data
gpio_bits_set	Set GPIO bits
gpio_bits_reset	Reset GPIO bits
gpio_bits_write	Write GPIO bits
gpio_port_write	Write GPIO ports
gpio_pin_wp_config	Configure GPIO pin write protection
gpio_event_output_config	Configure GPIO event output
gpio_event_output_enable	Enable/disable GPIO event output
gpio_pin_remap_config	Configure GPIO pin remapping
gpio_exint_line_config	Configure GPIO external interrupt lines

## 5.11.1 gpio\_reset function

The table below describes the function gpio\_reset.

**Table 212. gpio\_reset function**

Name	Description
Function name	gpio_reset
Function prototype	void gpio_reset(gpio_type *gpio_x);
Function description	GPIO is reset by CRM reset register
Input parameter	gpio_x: Select GPIO peripherals. This parameter can be G GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

**Example:**

```
gpio_reset(GPIOA);
```

## 5.11.2 gpio\_iomux\_reset function

The table below describes the function gpio\_iomux\_reset.

**Table 213. gpio\_iomux\_reset function**

Name	Description
Function name	gpio_iomux_reset
Function prototype	void gpio_iomux_reset();
Function description	IOMUX is reset by CRM reset register.
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

**Example:**

```
gpio_iomux_reset();
```

### 5.11.3 gpio\_init function

The table below describes the function gpio\_init.

**Table 214. gpio\_init function**

Name	Description
Function name	gpio_init
Function prototype	void gpio_init(gpio_type *gpio_x, gpio_init_type *gpio_init_struct);
Function description	Initialize GPIO peripherals
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG
Input parameter 2	gpio_init_struct: gpio_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### gpio\_init\_type structure

The gpio\_init\_type is defined in the at32f403\_gpio.h:

```
typedef struct
{
    uint32_t          gpio_pins;
    gpio_output_type  gpio_out_type;
    gpio_pull_type    gpio_pull;
    gpio_mode_type    gpio_mode;
    gpio_drive_type   gpio_drive_strength;
} gpio_init_type;
```

#### gpio\_pins

Select a GPIO pin.

GPIO_PINS_0:	GPIO pin 0
GPIO_PINS_1:	GPIO pin 1
GPIO_PINS_2:	GPIO pin 2
GPIO_PINS_3:	GPIO pin 3
GPIO_PINS_4:	GPIO pin 4
GPIO_PINS_5:	GPIO pin 5
GPIO_PINS_6:	GPIO pin 6
GPIO_PINS_7:	GPIO pin 7
GPIO_PINS_8:	GPIO pin 8
GPIO_PINS_9:	GPIO pin 9
GPIO_PINS_10:	GPIO pin 10
GPIO_PINS_11:	GPIO pin 11
GPIO_PINS_12:	GPIO pin 12
GPIO_PINS_13:	GPIO pin 13
GPIO_PINS_14:	GPIO pin 14
GPIO_PINS_15:	GPIO pin 15

#### gpio\_out\_type

Set GPIO output type.

GPIO_OUTPUT_PUSH_PULL:	GPIO push-pull
------------------------	----------------

GPIO\_OUTPUT\_OPEN\_DRAIN: GPIO open drain

#### gpio\_pull

Set GPIO pull-up or pull-down.

GPIO\_PULL\_NONE: No GPIO pull-up/pull-down

GPIO\_PULL\_UP: GPIO pull-up

GPIO\_PULL\_DOWN: GPIO pull-down

#### gpio\_mode

Set GPIO mode

GPIO\_MODE\_INPUT: GPIO input

GPIO\_MODE\_OUTPUT: GPIO output

GPIO\_MODE\_MUX: GPIO multiplexed mode

GPIO\_MODE\_ANALOG: GPIO analog mode

#### gpio\_drive\_strength

Set GPIO driver capability.

GPIO\_DRIVE\_STRENGTH\_STRONGER: Strong drive strength

GPIO\_DRIVE\_STRENGTH\_MODERATE: Moderate drive strength

#### Example:

```
gpio_init_type gpio_init_struct;
gpio_init_struct.gpio_pins = GPIO_PINS_0;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOA, &gpio_init_struct);
```

### 5.11.4 gpio\_default\_para\_init function

The table below describes the function gpio\_default\_para\_init.

**Table 215. gpio\_default\_para\_init function**

Name	Description
Function name	gpio_default_para_init
Function prototype	void gpio_default_para_init(gpio_init_type *gpio_init_struct);
Function description	Initialize GPIO default parameter
Input parameter	gpio_init_struct: gpio_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the gpio\_init\_struct.

**Table 216. gpio\_init\_struct default values**

Member	Default value
gpio_pins	GPIO_PINS_ALL
gpio_mode	GPIO_MODE_INPUT
gpio_out_type	GPIO_OUTPUT_PUSH_PULL
gpio_pull	GPIO_PULL_NONE
gpio_drive_strength	GPIO_DRIVE_STRENGTH_STRONGER

**Example:**

```
gpio_init_type gpio_init_struct;
gpio_default_para_init(&gpio_init_struct);
```

### 5.11.5 gpio\_input\_data\_bit\_read function

The table below describes the function gpio\_input\_data\_bit\_read.

**Table 217. gpio\_input\_data\_bit\_read function**

Name	Description
Function name	gpio_input_data_bit_read
Function prototype	flag_status gpio_input_data_bit_read(gpio_type *gpio_x, uint16_t pins);
Function description	Read GPIO input port pin
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG
Input parameter 2	Pins: indicates the GPIO pins, refer to <a href="#">gpio_pins</a> for details.
Output parameter	NA
Return value	Return GPIO input pin status
Required preconditions	NA
Called functions	NA

**Example:**

```
gpio_input_data_bit_read(GPIOA, GPIO_PINS_0);
```

### 5.11.6 gpio\_input\_data\_read function

The table below describes the function gpio\_input\_data\_read.

**Table 218. gpio\_input\_data\_read function**

Name	Description
Function name	gpio_input_data_read
Function prototype	uint16_t gpio_input_data_read(gpio_type *gpio_x);
Function description	Read GPIO input port
Input parameter	gpio_x: indicates the selected GPIO peripheral. This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG
Output parameter	NA
Return value	Return GPIO input port status
Required preconditions	NA
Called functions	NA

**Example:**

```
gpio_input_data_read(GPIOA);
```

## 5.11.7 gpio\_output\_data\_bit\_read function

The table below describes the function gpio\_output\_data\_bit\_read.

**Table 219. gpio\_output\_data\_bit\_read function**

Name	Description
Function name	gpio_output_data_bit_read
Function prototype	uint16_t gpio_output_data_bit_read(gpio_type *gpio_x);
Function description	Read GPIO output port pin
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG.
Input parameter 2	Pins: indicates the GPIO pins, refer to, refer to <a href="#">gpio_pins</a> for details.
Output parameter	NA
Return value	Return GPIO output pin status
Required preconditions	NA
Called functions	NA

**Example:**

```
gpio_output_data_bit_read(GPIOA, GPIO_PINS_0);
```

## 5.11.8 gpio\_output\_data\_read function

The table below describes the function gpio\_output\_data\_read.

**Table 220. gpio\_output\_data\_read function**

Name	Description
Function name	gpio_output_data_read
Function prototype	uint16_t gpio_output_data_read(gpio_type *gpio_x);
Function description	Read GPIO output port
Input parameter	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG.
Output parameter	NA
Return value	Read GPIO output port status
Required preconditions	NA
Called functions	NA

**Example:**

```
gpio_output_data_read(GPIOA);
```

## 5.11.9 gpio\_bits\_set function

The table below describes the function gpio\_bits\_set.

**Table 221. gpio\_bits\_set function**

Name	Description
Function name	gpio_bits_set
Function prototype	void gpio_bits_set(gpio_type *gpio_x, uint16_t pins);
Function description	Set GPIO pins
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG
Input parameter 2	Pins: indicates the GPIO pins, refer to <a href="#">gpio_pins</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
gpio_bits_set(GPIOA, GPIO_PINS_0);
```

## 5.11.10 gpio\_bits\_reset function

The table below describes the function gpio\_bits\_reset.

**Table 222. gpio\_bits\_reset function**

Name	Description
Function name	gpio_bits_reset
Function prototype	void gpio_bits_reset(gpio_type *gpio_x, uint16_t pins);
Function description	Reset GPIO pins
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG.
Input parameter 2	Pins: indicates the GPIO pins, refer to <a href="#">gpio_pins</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
gpio_bits_reset(GPIOA, GPIO_PINS_0);
```

### 5.11.11 gpio\_bits\_write function

The table below describes the function gpio\_bits\_write.

**Table 223. gpio\_bits\_write function**

Name	Description
Function name	gpio_bits_write
Function prototype	void gpio_bits_write(gpio_type *gpio_x, uint16_t pins, confirm_state bit_state);
Function description	Write GPIO pins
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG.
Input parameter 2	Pins: indicates the GPIO pins, refer to <a href="#">gpio_pins</a> for details.
Input parameter 3	bit_state: indicates GPIO pin value to write This parameter can be 1 (TRUE) or 0 (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
gpio_bits_write(GPIOA, GPIO_PINS_0, TRUE);
```

### 5.11.12 gpio\_port\_write function

The table below describes the function gpio\_port\_write.

**Table 224. gpio\_port\_write function**

Name	Description
Function name	gpio_port_write
Function prototype	void gpio_port_write(gpio_type *gpio_x, uint16_t port_value);
Function description	Write GPIO port
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG
Input parameter 2	port_value: indicates the port value to write This parameter can be 0x0000~0xFFFF.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
gpio_port_write(GPIOA, 0xFFFF);
```

### 5.11.13 gpio\_pin\_wp\_config function

The table below describes the function gpio\_pin\_wp\_config.

**Table 225. gpio\_pin\_wp\_config function**

Name	Description
Function name	gpio_pin_wp_config
Function prototype	void gpio_pin_wp_config(gpio_type *gpio_x, uint16_t pins);
Function description	Configure GPIO pin write protection
Input parameter 1	gpio_x: indicates the selected GPIO peripheral This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG
Input parameter 2	Pins: indicates the GPIO pins, refer to <a href="#">gpio_pins</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
gpio_pin_wp_config(GPIOA, GPIO_PINS_0);
```

### 5.11.14 gpio\_event\_output\_config function

The table below describes the function gpio\_event\_output\_config.

**Table 226. gpio\_event\_output\_config function**

Name	Description
Function name	gpio_event_output_config
Function prototype	void gpio_event_output_config(gpio_port_source_type gpio_port_source, gpio_pins_source_type gpio_pin_source);
Function description	Configure GPIO event output
Input parameter 1	gpio_port_source: indicates a GPIO port
Input parameter 2	gpio_pin_source: indicates a GPIO pin
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### gpio\_port\_source

Select a GPIO port.

GPIO\_PORT\_SOURCE\_GPIOA: GPIO port A  
GPIO\_PORT\_SOURCE\_GPIOB: GPIO port B  
GPIO\_PORT\_SOURCE\_GPIOC: GPIO port C  
GPIO\_PORT\_SOURCE\_GPIOD: GPIO port D  
GPIO\_PORT\_SOURCE\_GPIOE: GPIO port E  
GPIO\_PORT\_SOURCE\_GPIOF: GPIO port F  
GPIO\_PORT\_SOURCE\_GPIOG: GPIO port G

#### gpio\_pin\_source

Select a GPIO pin.

GPIO\_PINS\_SOURCE0: GPIO pin 0  
GPIO\_PINS\_SOURCE1: GPIO pin 1  
GPIO\_PINS\_SOURCE2: GPIO pin 2  
GPIO\_PINS\_SOURCE3: GPIO pin 3  
GPIO\_PINS\_SOURCE4: GPIO pin 4  
GPIO\_PINS\_SOURCE5: GPIO pin 5  
GPIO\_PINS\_SOURCE6: GPIO pin 6  
GPIO\_PINS\_SOURCE7: GPIO pin 7  
GPIO\_PINS\_SOURCE8: GPIO pin 8  
GPIO\_PINS\_SOURCE9: GPIO pin 9  
GPIO\_PINS\_SOURCE10: GPIO pin 10  
GPIO\_PINS\_SOURCE11: GPIO pin 11  
GPIO\_PINS\_SOURCE12: GPIO pin 12  
GPIO\_PINS\_SOURCE13: GPIO pin 13  
GPIO\_PINS\_SOURCE14: GPIO pin 14  
GPIO\_PINS\_SOURCE15: GPIO pin 15

#### Example:

```
gpio_event_output_config(GPIO_PORT_SOURCE_GPIOA, GPIO_PINS_SOURCE0);
```

## 5.11.15 gpio\_event\_output\_enable function

The table below describes the function gpio\_event\_output\_enable.

**Table 227. gpio\_event\_output\_enable function**

Name	Description
Function name	gpio_event_output_enable
Function prototype	void gpio_event_output_enable(confirm_state new_state);
Function description	Enable/disable GPIO event output
Input parameter	new_state: indicates GPIO event output status This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
gpio_event_output_enable(TRUE);
```

## 5.11.16 gpio\_pin\_remap\_config function

The table below describes the function gpio\_pin\_remap\_config.

**Table 228. gpio\_pin\_remap\_config function**

Name	Description
Function name	gpio_pin_remap_config
Function prototype	void gpio_pin_remap_config(uint32_t gpio_remap, confirm_state new_state);
Function description	Configure IOMUX pin
Input parameter 1	gpio_remap: indicates IOMUX peripherals
Input parameter 2	new_state: indicates IOMUX status This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**gpio\_remap**

Select IOMUX peripherals. It includes many parameters, to name a few here. For more information, please refer to the corresponding reference manual.

SPI1\_MUX\_01: spi1\_cs/i2s1\_ws(pa15), spi1\_sck/i2s1\_ck(pb3), spi1\_miso(pb4),  
spi1\_mosi/i2s1\_sd(pb5), i2s1\_mck(pb0)

SPI1\_MUX\_10: spi1\_cs/i2s1\_ws(pa4), spi1\_sck/i2s1\_ck(pa5), spi1\_miso(pa6),  
spi1\_mosi/i2s1\_sd(pa7), i2s1\_mck(pb6)

.....

SDIO2\_MUX11: sdio2\_ck(pa2), sdio2\_cmd(pa3), sdio2\_d0(pa4), sdio2\_d1(pa5), sdio2\_d2(pa6),  
sdio2\_d3(pa7)

EXT\_SPIM\_EN\_MUX: enable external SPI Flash interface

**Example:**

```
gpio_pin_remap_config(SPI1_MUX_01, TRUE);
```

### 5.11.17 gpio\_exint\_line\_config function

The table below describes the function gpio\_exint\_line\_config.

**Table 229. gpio\_exint\_line\_config function**

Name	Description
Function name	gpio_exint_line_config
Function prototype	void gpio_exint_line_config(gpio_port_source_type gpio_port_source, gpio_pins_source_type gpio_pin_source);
Function description	Config GPIO external interrupt lines
Input parameter 1	gpio_port_source: GPIO port selection
Input parameter 2	gpio_pin_source: GPIO pin selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### gpio\_port\_source

For GPIO port selection, refer to [gpio\\_port\\_source](#) for details.

#### gpio\_pin\_source

For GPIO pin selection, refer to [gpio\\_pin\\_source](#) for details.

#### Example:

```
gpio_exint_line_config(GPIO_PORT_SOURCE_GPIOA, GPIO_PINS_SOURCE0);
```

## 5.12 I2C interfaces

The I2C register structure i2c\_type is defined in the “at32f403\_i2c.h”:

```
/*
 * @brief type define i2c register all
 */
typedef struct
{
} i2c_type;
```

Table 255 gives a list of the I2C registers.

**Table 230. Summary of I2C registers**

Register	Description
ctrl1	I2C control register 1
ctrl2	I2C control register 2
oaddr1	I2C own address register 1
oaddr2	I2C own address register 2
dt	I2C data register
sts1	I2C status register 1
sts2	I2C status register 2
clkctrl	I2C clock control register
tmrise	I2C clock rise register

The table below gives a list of I2C library functions.

**Table 231. Summary of I2C library functions**

Function name	Description
i2c_reset	I2C peripheral reset
i2c_software_reset	I2C software reset
i2c_init	Set I2C bus speed
i2c_own_address1_set	Set I2C own address 1
i2c_own_address2_set	Set I2C own address 2
i2c_own_address2_enable	Enable I2C own address 2
i2c_smbus_enable	Enable Smbus mode
i2c_enable	Enable I2C
i2c_fast_mode_duty_set	Set fast mode duty cycle
i2c_clock_stretch_enable	Enable clock stretching capability
i2c_ack_enable	Enable ACK response
i2c_master_receive_ack_set	Set master receive mode ACK response
i2c_pec_position_set	Set PEC location in Smbus mod and master receive mode
i2c_general_call_enable	Enable general call (broadcast address enable)
i2c_arp_mode_enable	Enable SMBus ARP address
i2c_smbus_mode_set	SMBus device mode selection
i2c_smbus_alert_set	Set SMBus alert pin level

i2c_pec_transmit_enable	Enable PEC transmit
i2c_pec_calculate_enable	Enable PEC calculation
i2c_pec_value_get	Get current PEC value
i2c_dma_end_transfer_set	DMA transfer end indication
i2c_dma_enable	Enable DMA transfer
i2c_interrupt_enable	Enable I2C interrupts
i2c_start_generate	Generate Start condition
i2c_stop_generate	Generate Stop condition
i2c_7bit_address_send	Send 7-bit slave address
i2c_data_send	Send data
i2c_data_receive	Receive data
i2c_flag_get	Get flag
i2c_flag_clear	Clear flag

**Table 232. I2C application-layer library functions**

Function name	Description
i2c_config	I2C application initialization
i2c_lowlevel_init	I2C low-layer initialization
i2c_wait_end	I2C wait data transmit complete
i2c_wait_flag	I2C wait flag
i2c_master_transmit	I2C master transmits data (polling mode)
i2c_master_receive	I2C master receives data (polling mode)
i2c_slave_transmit	I2C slave transmits data (polling mode)
i2c_slave_receive	I2C slave receives data (polling mode)
i2c_master_transmit_int	I2C master transmits data (interrupt mode)
i2c_master_receive_int	I2C master receives data (interrupt mode)
i2c_slave_transmit_int	I2C slave transmits data (interrupt mode)
i2c_slave_receive_int	I2C slave receives data (interrupt mode)
i2c_master_transmit_dma	I2C master transmits data (DMA mode)
i2c_master_receive_dma	I2C master receives data (DMA mode)
i2c_slave_transmit_dma	I2C slave transmits data (DMA mode)
i2c_slave_receive_dma	I2C slave receives data (DMA mode)
i2c_memory_write	I2C writes data to EEPROM (polling mode)
i2c_memory_write_int	I2C writes data to EEPROM (interrupt mode)
i2c_memory_write_dma	I2C writes data to EEPROM (DMA mode)
i2c_memory_read	I2C reads from EEPROM (polling mode)
i2c_memory_read_int	I2C reads from EEPROM (interrupt mode)
i2c_memory_read_dma	I2C reads from EEPROM (DMA mode)
i2c_evt_irq_handler	I2C event interrupt function
i2c_err_irq_handler	I2C error interrupt function
i2c_dma_tx_irq_handler	I2C DMA Tx interrupt function
i2c_dma_rx_irq_handler	I2C DMA Rx interrupt function

### 5.12.1 i2c\_reset function

The table below describes the function i2c\_reset.

**Table 233. i2c\_reset function**

Name	Description
Function name	i2c_reset
Function prototype	void i2c_reset(i2c_type *i2c_x)
Function description	Reset all I2C registers to their initial values through CRM (Clock and reset management)
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void crm_periph_reset(crm_periph_reset_type value, confirm_state new_state);

**Example:**

```
i2c_reset(I2C1);
```

### 5.12.2 i2c\_software\_reset function

The table below describes the function i2c\_software\_reset.

**Table 234. i2c\_software\_reset function**

Name	Description
Function name	i2c_software_reset
Function prototype	void i2c_software_reset(i2c_type *i2c_x, confirm_state new_state);
Function description	Reset I2C by software, in the same way of the function i2c_reset(i2c_type *i2c_x)
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3
Input parameter 2	new_state: indicates software reset status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_software_reset(I2C1, TRUE);
i2c_software_reset(I2C1, FALSE);
```

### 5.12.3 i2c\_init function

The table below describes the function i2c\_init.

**Table 235. i2c\_init function**

Name	Description
Function name	i2c_init
Function prototype	void i2c_init(i2c_type *i2c_x, i2c_fsmode_duty_cycle_type duty, uint32_t speed);
Function description	Set I2C bus speed and duty cycle in fast mode
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3
Input parameter 2	Duty: SCL bus duty cycle in fast mode Refer to the “duty” description below for details.
Input parameter 3	Speed: bus speed in Hz
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### duty

SCL duty cycle in fast mode ( $\geq 400\text{kHz}$ )

I2C\_FSMODE\_DUTY\_2\_1: SCL duty cycle is 2: 1

I2C\_FSMODE\_DUTY\_16\_9: SCL duty cycle is 16: 9

#### Example:

```
i2c_init(I2C1, I2C_FSMODE_DUTY_2_1, 100000);
```

### 5.12.4 i2c\_own\_address1\_set function

The table below describes the function i2c\_own\_address1\_set.

**Table 236. i2c\_own\_address1\_set function**

Name	Description
Function name	i2c_own_address1_set
Function prototype	void i2c_own_address1_set(i2c_type *i2c_x, i2c_address_mode_type mode, uint16_t address);
Function description	Set own address 1
Input parameter 1	Mode: Own address 1 address mode Refer to the “mode” description below for details.
Input parameter 2	Address: own address 1
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### mode

Own address 1 address mode

I2C\_ADDRESS\_MODE\_7BIT: 7-bit address

I2C\_ADDRESS\_MODE\_10BIT: 10-bit address

#### Example:

```
i2c_own_address1_set(I2C1, I2C_ADDRESS_MODE_7BIT, 0xA0);
```

## 5.12.5 i2c\_own\_address2\_set function

The table below describes the function i2c\_own\_address2\_set.

**Table 237. i2c\_own\_address2\_set function**

Name	Description
Function name	i2c_own_address2_set
Function prototype	void i2c_own_address2_set(i2c_type *i2c_x, uint8_t address);
Function description	Set own address 2. The address 2 becomes active only after it is enabled. Note: only 7-bit address is supported, not 10-bit address mode
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	Address: own address 2
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

i2c_own_address2_set(I2C1, 0xB0);
-----------------------------------

## 5.12.6 i2c\_own\_address2\_enable function

The table below describes the function i2c\_own\_address2\_enable.

**Table 238. i2c\_own\_address2\_enable function**

Name	Description
Function name	i2c_own_address2_enable
Function prototype	void i2c_own_address2_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable own address 2. The address becomes active only after it is enabled. Note that this function should be used in conjunction with the i2c_own_address2_set.
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	new_state: indicates address 2 status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

i2c_own_address2_enable(I2C1, TRUE);
--------------------------------------

## 5.12.7 i2c\_smbus\_enable function

The table below describes the function i2c\_smbus\_enable.

**Table 239. i2c\_smbus\_enable function**

Name	Description
Function name	i2c_smbus_enable
Function prototype	void i2c_smbus_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable SMBus mode. After power-on reset, it is I2C mode by default.
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	new_state: indicates SMBus mode status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

i2c_smbus_enable(I2C1, TRUE);
-------------------------------

## 5.12.8 i2c\_enable function

The table below describes the function i2c\_enable.

**Table 240. i2c\_enable function**

Name	Description
Function name	i2c_enable
Function prototype	void i2c_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable I2C peripheral
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	new_state: indicates I2C status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

i2c_enable(I2C1, TRUE);
-------------------------

### 5.12.9 i2c\_fast\_mode\_duty\_set function

The table below describes the function i2c\_fast\_mode\_duty\_set.

**Table 241. i2c\_fast\_mode\_duty\_set function**

Name	Description
Function name	i2c_fast_mode_duty_set
Function prototype	void i2c_fast_mode_duty_set(i2c_type *i2c_x, i2c_fsmode_duty_cycle_type duty);
Function description	Configure the ratio of SCL low level to high level in fast mode. This function works in the same way of the duty parameter in the void i2c_init(i2c_type *i2c_x, i2c_fsmode_duty_cycle_type duty, uint32_t speed).
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	Duty: indicates the duty cycle of SCL in fast mode Refer to the “duty” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### duty

SCL duty cycle in fast mode ( $\geq 400\text{kHz}$ ).

I2C\_FSMODE\_DUTY\_2\_1: SCL duty cycle is 2: 1

I2C\_FSMODE\_DUTY\_16\_9: SCL duty cycle is 16: 9

#### Example:

```
i2c_fast_mode_duty_set(I2C1, I2C_FSMODE_DUTY_2_1);
```

### 5.12.10 i2c\_clock\_stretch\_enable function

The table below describes the function i2c\_clock\_stretch\_enable.

**Table 242. i2c\_clock\_stretch\_enable function**

Name	Description
Function name	i2c_clock_stretch_enable
Function prototype	void i2c_clock_stretch_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable clock stretching capability. This function is applicable to slave mode only. In most cases, enabling the clock stretching mode is recommended in order to prevent slave from having no sufficient time to receive or send data due to slow process speed, which causes a loss of data. It should be noted that the host must be able to support clock stretching function before using this mode by slave. For example, some hosts based on IO analog are not equipped with the clock stretching capability.
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	new_state: indicates clock stretching status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

i2c_clock_stretch_enable(I2C1, TRUE);
---------------------------------------

### 5.12.11 i2c\_ack\_enable function

The table below describes the function i2c\_ack\_enable.

**Table 243. i2c\_ack\_enable function**

Name	Description
Function name	i2c_ack_enable
Function prototype	void i2c_ack_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	<p>Enable ACK and NACK.</p> <p>This function is used to enable ACK or NACK of each byte in master and slave mode. For ACK information on I2C communication protocol, refer to I2C protocol or AT32 reference manual.</p>
Input parameter 1	<p>i2c_x: indicates the selected I2C peripheral</p> <p>This parameter can be I2C1, I2C2, I2C3.</p>
Input parameter 2	<p>new_state: indicates ACK response status</p> <p>This parameter can be TRUE or FALSE.</p>
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

i2c_ack_enable(I2C1, TRUE);
-----------------------------

### 5.12.12 i2c\_master\_receive\_ack\_set function

The table below describes the function i2c\_master\_receive\_ack\_set.

**Table 244. i2c\_master\_receive\_ack\_set function**

Name	Description
Function name	i2c_master_receive_ack_set
Function prototype	void i2c_master_receive_ack_set(i2c_type *i2c_x, i2c_master_ack_type pos)
Function description	<p>Enable master receive ACK response. This function is used in master receive mode to define the location where the function void i2c_ack_enable(i2c_type *i2c_x, confirm_state new_state) becomes active. The function is aimed at returning a correct NACK response while two bytes are received in master receive mode.</p>
Input parameter 1	<p>i2c_x: indicates the selected I2C peripheral</p> <p>This parameter can be I2C1, I2C2, I2C3.</p>
Input parameter 2	<p>Pos: indicates the location of ACKEN</p> <p>Refer to the "pos" description below for details.</p>
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**pos**

ACKEN valid position.

I2C\_MASTER\_ACK\_CURRENT: ACKEN bit effective on the current byte being transferred

I2C\_MASTER\_ACK\_NEXT: ACKEN bit effective on the next byte to be transferred

**Example:**

```
i2c_master_receive_ack_set(I2C1, TRUE);
```

### 5.12.13 i2c\_pec\_position\_set function

The table below describes the function i2c\_pec\_position\_set.

**Table 245. i2c\_pec\_position\_set function**

Name	Description
Function name	i2c_pec_position_set
Function prototype	void i2c_pec_position_set(i2c_type *i2c_x, i2c_pec_position_type pos);
Function description	Set PEC location in SMBus and master receive mode. This function is used to receive PEC and return NACK when two bytes are received in master receive mode.
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	Pos: PEC position Refer to the “pos” description below fore details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**pos**

PEC valid position.

I2C\_PEC\_POSITION\_CURRENT: Current byte is PEC

I2C\_PEC\_POSITION\_NEXT: Next byte is PEC

**Example:**

```
i2c_pec_position_set(I2C1, I2C_PEC_POSITION_CURRENT);
```

## 5.12.14 i2c\_general\_call\_enable function

The table below describes the function i2c\_general\_call\_enable.

**Table 246. i2c\_general\_call\_enable function**

Name	Description
Function name	i2c_general_call_enable
Function prototype	void i2c_general_call_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable general call. After enabled, address 0x00 is ACKed or NACKed.
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	new_state: indicates general call status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_general_call_enable(I2C1, TRUE);
```

## 5.12.15 i2c\_arp\_mode\_enable function

The table below describes the function i2c\_arp\_mode\_enable.

**Table 247. i2c\_arp\_mode\_enable function**

Name	Description
Function name	i2c_arp_mode_enable
Function prototype	void i2c_arp_mode_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable SMBus ARP In SMBus master mode: respond to master address 0001000x In SMBus device mode: respond to device default address 0001100x For more information on ARP, refer to SMBUS protocol.
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	new_state: indicates ARP address status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_arp_mode_enable(I2C1, TRUE);
```

## 5.12.16 i2c\_smbus\_mode\_set function

The table below describes the function i2c\_smbus\_mode\_set.

**Table 248. i2c\_smbus\_mode\_set function**

Name	Description
Function name	i2c_smbus_mode_set
Function prototype	void i2c_smbus_mode_set(i2c_type *i2c_x, i2c_smbus_mode_set_type mode);
Function description	Select SMBus device mode, including SMBus host or SMBus device
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	Mode: SMBus device mode Refer to the “mode” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### mode

SMBus device mode.

I2C\_SMBUS\_MODE\_DEVICE: SMBus device

I2C\_SMBUS\_MODE\_HOST: SMBus host

### Example:

```
i2c_smbus_mode_set(I2C1, I2C_SMBUS_MODE_HOST);
```

## 5.12.17 i2c\_smbus\_alert\_set function

The table below describes the function i2c\_smbus\_alert\_set.

**Table 249. i2c\_smbus\_alert\_set function**

Name	Description
Function name	i2c_smbus_alert_set
Function prototype	void i2c_smbus_alert_set(i2c_type *i2c_x, i2c_smbus_alert_set_type level);
Function description	Set SMBus alert pin low or high
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	Level: SMBus alert pin level Refer to the “level” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### level

SMBus alert pin level

I2C\_SMBUS\_ALERT\_LOW: SMBus alert pin low

I2C\_SMBUS\_ALERT\_HIGH: SMBus alert pin high

### Example:

```
i2c_smbus_alert_set(I2C1, I2C_SMBUS_ALERT_LOW);
```

### 5.12.18 i2c\_pec\_transmit\_enable function

The table below describes the function i2c\_pec\_transmit\_enable.

**Table 250. i2c\_pec\_transmit\_enable function**

Name	Description
Function name	i2c_pec_transmit_enable
Function prototype	void i2c_pec_transmit_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable PEC transmit (send or receive PEC). After calling this function, the PEC is ready to send or receive.
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	new_state: PEC transfer status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_pec_transmit_enable(I2C1, TRUE);
```

### 5.12.19 i2c\_pec\_calculate\_enable function

The table below describes the function i2c\_pec\_calculate\_enable.

**Table 251. i2c\_pec\_calculate\_enable function**

Name	Description
Function name	i2c_pec_calculate_enable
Function prototype	void i2c_pec_calculate_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable PEC calculation
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	new_state: PEC calculation status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_pec_calculate_enable(I2C1, TRUE);
```

## 5.12.20 i2c\_pec\_value\_get function

The table below describes the function i2c\_pec\_value\_get.

**Table 252. i2c\_pec\_value\_get function**

Name	Description
Function name	i2c_pec_value_get
Function prototype	uint8_t i2c_pec_value_get(i2c_type *i2c_x);
Function description	Get PEC value
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Output parameter	uint8_t: current PEC value
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
Pec_value = i2c_pec_value_get(I2C1);
```

## 5.12.21 i2c\_dma\_end\_transfer\_set function

The table below describes the function i2c\_dma\_end\_transfer\_set.

**Table 253. i2c\_dma\_end\_transfer\_set function**

Name	Description
Function name	i2c_dma_end_transfer_set
Function prototype	void i2c_dma_end_transfer_set(i2c_type *i2c_x, confirm_state new_state);
Function description	Indicates DMA transfer complete, that is, indicating whether the last data is being sent or not.
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	new_state: indicates whether it is the last data being transferred or not. This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_dma_end_transfer_set(I2C1, TRUE);
```

## 5.12.22 i2c\_dma\_enable function

The table below describes the function i2c\_dma\_enable.

**Table 254. i2c\_dma\_enable function**

Name	Description
Function name	i2c_dma_enable
Function prototype	void i2c_dma_enable(i2c_type *i2c_x, confirm_state new_state);
Function description	Enable DMA transfer
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	new_state: DMA status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

i2c_dma_enable(I2C1, TRUE);
-----------------------------

## 5.12.23 i2c\_interrupt\_enable function

The table below describes the function i2c\_interrupt\_enable.

**Table 255. i2c\_interrupt\_enable function**

Name	Description
Function name	i2c_interrupt_enable
Function prototype	void i2c_interrupt_enable(i2c_type *i2c_x, uint16_t source, confirm_state new_state)
Function description	Enable I2C interrupts
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	Source: Interrupt source Refer to the "source" description below for details.
Input parameter 3	new_state: interrupt status This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**source**

Interrupt sources.

I2C\_DATA\_INT: Data interrupt

I2C\_EV\_INT: Event interrupt

I2C\_ERR\_INT: Error interrupt

**Example:**

i2c_interrupt_enable(I2C1, I2C_DATA_INT, TRUE);
---

### 5.12.24 i2c\_start\_generate function

The table below describes the function i2c\_start\_generate.

**Table 256. i2c\_start\_generate function**

Name	Description
Function name	i2c_start_generate
Function prototype	void i2c_start_generate(i2c_type *i2c_x);
Function description	Generate Start condition (for host)
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_start_generate(I2C1);
```

### 5.12.25 i2c\_stop\_generate function

The table below describes the function i2c\_stop\_generate.

**Table 257. i2c\_stop\_generate function**

Name	Description
Function name	i2c_stop_generate
Function prototype	void i2c_stop_generate(i2c_type *i2c_x);
Function description	Generate Stop condition
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_stop_generate(I2C1);
```

## 5.12.26 i2c\_7bit\_address\_send function

The table below describes the function i2c\_7bit\_address\_send.

**Table 258. i2c\_7bit\_address\_send function**

Name	Description
Function name	i2c_7bit_address_send
Function prototype	void i2c_7bit_address_send(i2c_type *i2c_x, uint8_t address, i2c_direction_type direction);
Function description	Send 7-bit slave address (for host)
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	Address: slave address
Input parameter 3	Direction: data transfer direction Refer to the "direction" description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### direction

Data transfer direction.

I2C\_DIRECTION\_TRANSMIT: Master transmit

I2C\_DIRECTION\_RECEIVE: Master receive

### Example:

```
i2c_7bit_address_send(I2C1, 0xB0, I2C_DIRECTION_TRANSMIT);
```

## 5.12.27 i2c\_data\_send function

The table below describes the function i2c\_data\_send.

**Table 259. i2c\_data\_send function**

Name	Description
Function name	i2c_data_send
Function prototype	void i2c_data_send(i2c_type *i2c_x, uint8_t data);
Function description	Send data
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	Data: send data
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### Example:

```
i2c_data_send(I2C1, 0x55);
```

## 5.12.28 i2c\_data\_receive function

The table below describes the function i2c\_data\_receive.

**Table 260. i2c\_data\_receive function**

Name	Description
Function name	i2c_data_receive
Function prototype	uint8_t i2c_data_receive(i2c_type *i2c_x);
Function description	Receive data
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Output parameter	uint8_t: receive data
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
data_value = i2c_data_receive(I2C1);
```

## 5.12.29 i2c\_flag\_get function

The table below describes the function i2c\_flag\_get.

**Table 261. i2c\_flag\_get function**

Name	Description
Function name	i2c_flag_get
Function prototype	flag_status i2c_flag_get(i2c_type *i2c_x, uint32_t flag);
Function description	Get flag status
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	Flag: Flag selection Refer to the "flag" description below for details.
Output parameter	NA
Return value	flag_status: indicates Flag status Returns SET or RESET
Required preconditions	NA
Called functions	NA

### flag

This is used for flag selection, including:

I2C_STARTF_FLAG:	Start condition generation complete flag
I2C_ADDR7F_FLAG:	0~7 bit address match flag
I2C_TDC_FLAG:	Data transfer complete flag
I2C_ADDRHF_FLAG:	9~8 bit address head match flag (host)
I2C_STOPF_FLAG:	Stop condition generation complete flag
I2C_RDBF_FLAG:	Receive data buffer full flag
I2C_TDBE_FLAG:	Transmit data buffer empty flag
I2C_BUSERR_FLAG:	Bus error flag
I2C_ARLOST_FLAG:	Arbitration lost flag

I2C_ACKFAIL_FLAG:	Acknowledge failure flag
I2C_OUF_FLAG:	Overflow flag
I2C_PECERR_FLAG:	PEC receive error flag
I2C_TMOUT_FLAG:	SMBus timeout flag
I2C_ALERTF_FLAG:	SMBus alert flag
I2C_TRMODE_FLAG:	Transfer mode
I2C_BUSYF_FLAG:	Bus busy flag
I2C_DIRF_FLAG:	Transmission direction flag
I2C_GCADDRF_FLAG:	General call address reception flag
I2C_DEVADDRF_FLAG:	SMBus device address reception flag
I2C_HOSTADDRF_FLAG:	SMBus host address reception flag
I2C_ADDR2_FLAG:	Received address 2 flag

**Example:**

```
i2c_flag_get(I2C1, I2C_STARTF_FLAG);
```

### 5.12.30 i2c\_interrupt\_flag\_get function

The table below describes the function i2c\_interrupt\_flag\_get.

**Table 262. i2c\_interrupt\_flag\_get function**

Name	Description
Function name	i2c_interrupt_flag_get
Function prototype	flag_status i2c_interrupt_flag_get(i2c_type *i2c_x, uint32_t flag);
Function description	Get flag status and judge the corresponding interrupt enable bit
Input parameter 1	i2c_x indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	flag: flag selection Refer to the "flag" description below for details.
Output parameter	NA
Return value	flag_status: indicates Flag status Returns SET or RESET
Required preconditions	NA
Called functions	NA

#### flag

This is used for flag selection, including:

I2C_STARTF_FLAG:	Start condition generation complete flag
I2C_ADDR7F_FLAG:	0~7 bit address match flag
I2C_TDC_FLAG:	Data transfer complete flag
I2C_ADDRHF_FLAG:	9~8 bit address head match flag (host)
I2C_STOPF_FLAG:	Stop condition generation complete flag
I2C_RDBF_FLAG:	Receive data buffer full flag
I2C_TDBE_FLAG:	Transmit data buffer full flag
I2C_BUSERR_FLAG:	Bus error flag
I2C_ARLOST_FLAG:	Arbitration lost flag
I2C_ACKFAIL_FLAG:	Acknowledge failure flag
I2C_OUF_FLAG:	Overflow flag
I2C_PECERR_FLAG:	PEC receive error flag
I2C_TMOUT_FLAG:	SMBus timeout flag

I2C\_ALERTF\_FLAG: SMBus alert flag

**Example**

```
i2c_interrupt_flag_get(I2C1, I2C_STARTF_FLAG);
```

### 5.12.31 i2c\_flag\_clear function

The table below describes the function i2c\_flag\_clear.

**Table 263. i2c\_flag\_clear function**

Name	Description
Function name	i2c_flag_clear
Function prototype	void i2c_flag_clear(i2c_type *i2c_x, uint32_t flag);
Function description	Clear flag
Input parameter 1	i2c_x: indicates the selected I2C peripheral This parameter can be I2C1, I2C2, I2C3.
Input parameter 2	Flag: indicates the selected flag Refer to the "flag" description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**flag**

This is used for flag selection, including:

I2C\_BUSERR\_FLAG: Bus error flag

I2C\_ARLOST\_FLAG: Arbitration lost flag

I2C\_ACKFAIL\_FLAG: Acknowledge failure flag

I2C\_OUF\_FLAG: Overflow flag

I2C\_PECERR\_FLAG: PEC receive error flag

I2C\_TMOUT\_FLAG: SMBus timeout flag

I2C\_ALERTF\_FLAG: SMBus alert flag

I2C\_ADDR7F\_FLAG: 0~7 bit address match flag

I2C\_STOPF\_FLAG: Stop condition generation complete flag

**Example:**

```
i2c_flag_clear(I2C1, I2C_ACKFAIL_FLAG);
```

## 5.12.32 i2c\_config function

The table below describes the function i2c\_config.

**Table 264. i2c\_config function**

Name	Description
Function name	i2c_config
Function prototype	void i2c_config(i2c_handle_type* hi2c);
Function description	Initialize I2C. The function i2c_lowlevel_init() is called to initialize I2C, GPIO, DMA, interrupts, and so on.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void i2c_lowlevel_init(i2c_handle_type* hi2c);

### i2c\_handle\_type\* hi2c

The i2c\_handle\_type is defined in the i2c\_application.h:

```
typedef struct
{
    i2c_type          *i2cx;
    uint8_t            *pbuf;
    __IO uint16_t      pcount;
    __IO uint32_t      mode;
    __IO uint32_t      timeout;
    __IO uint32_t      status;
    __IO i2c_status_type error_code;
    dma_channel_type  *dma_tx_channel;
    dma_channel_type  *dma_rx_channel;
    dma_init_type      dma_init_struct;
}i2c_handle_type;
```

#### i2cx

Indicates the selected I2C peripheral, and this parameter can be I2C1 or I2C2.

#### pbuf

array of receive/send data

#### pcount

number of receive/send data

#### mode

I2C communication mode for internal state machine, users don't need to care about

#### timeout

communication timeout

#### status

Transfer status for internal state machine, users don't need to care about

#### error\_code

Enumerate error code of the i2c\_status\_type. It records the erroneous code when an error occurs during communication.

I2C_OK:	Communication OK
I2C_ERR_STEP_1:	Step 1 error
I2C_ERR_STEP_2:	Step 2 error
I2C_ERR_STEP_3:	Step 3 error
I2C_ERR_STEP_4:	Step 4 error
I2C_ERR_STEP_5:	Step 5 error
I2C_ERR_STEP_6:	Step 6 error
I2C_ERR_STEP_7:	Step 7 error
I2C_ERR_STEP_8:	Step 8 error
I2C_ERR_STEP_9:	Step 9 error
I2C_ERR_STEP_10:	Step 10 error
I2C_ERR_STEP_11:	Step 11 error
I2C_ERR_STEP_12:	Step 12 error
I2C_ERR_START:	START condition error
I2C_ERR_ADDR10:	10-bit address header (bit9~8) error
I2C_ERR_ADDR:	Address error
I2C_ERR_STOP:	STOP condition error
I2C_ERR_ACKFAIL:	Acknowledge error
I2C_ERR_TIMEOUT:	Timeout
I2C_ERR_INTERRUPT:	Error event occurred and error interrupt entry

**dma\_tx\_channel**

I2C Tx DMA channel

**dma\_rx\_channel**

I2C Rx DMA channel

**dma\_init\_struct**

DMA initialization structure

**Example:**

```
i2c_handle_type hi2c;  
hi2c.i2cx = I2C1;  
i2c_config(&hi2c);
```

### 5.12.33 i2c\_lowlevel\_init function

The table below describes the function i2c\_lowlevel\_init.

Table 265. i2c\_lowlevel\_init function

Name	Description
Function name	i2c_lowlevel_init
Function prototype	void i2c_lowlevel_init(i2c_handle_type* hi2c);
Function description	I2C low-layer initialization function. It is used in the function i2c_config to initialize I2C, GPIO, DMA, interrupts and so on. The I2C initialization needs to be done in the function.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
void i2c_lowlevel_init(i2c_handle_type* hi2c)
{
    if(hi2c->i2cx == I2C1)
    {
        Initialize I2C1
    }
    else if(hi2c->i2cx == I2C2)
    {
        Initialize I2C2
    }
}
```

### 5.12.34 i2c\_wait\_end function

The table below describes the function i2c\_wait\_end.

**Table 266. i2c\_wait\_end function**

Name	Description
Function name	i2c_wait_end
Function prototype	i2c_status_type i2c_wait_end(i2c_handle_type* hi2c, uint32_t timeout);
Function description	Wait for the completion of communication. This function is used in DMA and interrupt transfer mode as both of them are non-blocking and can be used to wait for the end of transfer.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	Timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
if (i2c_master_transmit_dma(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF) != I2C_OK)
{
    error_handler(i2c_status);
}

/* Wait for the end of communication */
if(i2c_wait_end(&hi2c, 0xFFFFFFFF) != I2C_OK)
{
    error_handler(i2c_status);
}
```

### 5.12.35 i2c\_wait\_flag function

The table below describes the function i2c\_wait\_flag.

**Table 267. i2c\_wait\_flag function**

Name	Description
Function name	i2c_wait_flag
Function prototype	i2c_status_type i2c_wait_flag(i2c_handle_type* hi2c, uint32_t flag, uint32_t event_check, uint32_t timeout)
Function description	Wait for the flag to set or reset Users need to wait for BUSYF flag to reset, and other flags except for BUSYF to set.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">section 5.12.32</a> for details.
Input parameter 2	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 3	event_check: check whether an event is generated or not while waiting for flags Refer to the “event_check” description below for details.
Input parameter 4	Timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

#### flag

Flag selection.

I2C_STARTF_FLAG:	Start condition generation complete flag
I2C_ADDR7F_FLAG:	0~7 bit address match flag
I2C_TDC_FLAG:	Data transfer complete flag
I2C_ADDRHF_FLAG:	9~8 bit address head match flag (host)
I2C_STOPF_FLAG:	Stop condition generation complete flag
I2C_RDBF_FLAG:	Receive data buffer full flag
I2C_TDBE_FLAG:	Transmit data buffer empty flag
I2C_BUSERR_FLAG:	Bus error flag
I2C_ARLOST_FLAG:	Arbitration lost flag
I2C_ACKFAIL_FLAG:	Acknowledge failure flag
I2C_OUF_FLAG:	Overflow flag
I2C_PECERR_FLAG:	PEC receive error flag
I2C_TMOUT_FLAG:	SMBus timeout flag
I2C_ALERTF_FLAG:	SMBus alert flag
I2C_TRMODE_FLAG:	Transfer mode
I2C_BUSYF_FLAG:	Bus busy flag
I2C_DIRF_FLAG:	Transmission direction flag
I2C_GCADDRF_FLAG:	General call address reception flag
I2C_DEVADDRF_FLAG:	SMBus device address reception flag
I2C_HOSTADDRF_FLAG:	SMBus host address reception flag
I2C_ADDR2_FLAG:	Received address 2 flag

**event\_check**

While waiting for a flag, check if an event occurred or not.

I2C\_EVENT\_CHECK\_NONE: None event check

I2C\_EVENT\_CHECK\_ACKFAIL: ACKFAIL event check

I2C\_EVENT\_CHECK\_STOP: STOP event check

**Example:**

```
i2c_wait_flag(&hi2c, I2C_BUSYF_FLAG, I2C_EVENT_CHECK_NONE, 0xFFFFFFFF);
```

### 5.12.36 i2c\_master\_transmit function

The table below describes the function i2c\_master\_transmit.

**Table 268. i2c\_master\_transmit function**

Name	Description
Function name	i2c_master_transmit
Function prototype	i2c_status_type i2c_master_transmit(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master transmits data (polling mode). This function is in blocking mode, which means that I2C transfer comes to end after the execution of this function.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	Address: slave address
Input parameter 3	Pdata: array address of data to send
Input parameter 4	Size: number of data transmit
Input parameter 5	Timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_master_transmit(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.37 i2c\_master\_receive function

The table below describes the function i2c\_master\_receive.

**Table 269. i2c\_master\_receive function**

Name	Description
Function name	i2c_master_receive
Function prototype	i2c_status_type i2c_master_receive(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master receives data (polling mode). This function is in blocking mode, which means that I2C transfer comes to end after the execution of this function
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	address: slave address
Input parameter 3	pdata: array address of data to send
Input parameter 4	Size: number of data received
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_master_receive(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.38 i2c\_slave\_transmit function

The table below describes the function i2c\_slave\_transmit.

**Table 270. i2c\_slave\_transmit function**

Name	Description
Function name	i2c_slave_transmit
Function prototype	i2c_status_type i2c_slave_transmit(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave sends data (polling mode). This function is in blocking mode, which means that I2C transfer comes to end after the execution of this function
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	pdata: array address of data to send
Input parameter 3	size: number of data transmit
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_slave_transmit(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.39 i2c\_slave\_receive function

The table below describes the function i2c\_slave\_receive.

**Table 271. i2c\_slave\_receive function**

Name	Description
Function name	i2c_slave_receive
Function prototype	i2c_status_type i2c_slave_receive(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave receives data (polling mode). This function is in blocking mode, which means that I2C transfer comes to end after the execution of this function
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	pdata: array address of data to send
Input parameter 3	Size: number of data received
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_slave_receive(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.40 i2c\_master\_transmit\_int function

The table below describes the function i2c\_master\_transmit\_int.

**Table 272. i2c\_master\_transmit\_int function**

Name	Description
Function name	i2c_master_transmit_int
Function prototype	i2c_status_type i2c_master_transmit_int(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master sends data (interrupt mode). This function is non-blocking, which means that I2C communication has yet to complete after the execution of this function. It is possible to wait for the end of communication by calling the function i2c_wait_end().
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	Address: slave address
Input parameter 3	pdata: array address of data to send
Input parameter 4	Size: number of data transmit
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_master_transmit_int(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.41 i2c\_master\_receive\_int function

The table below describes the function i2c\_master\_receive\_int.

**Table 273. i2c\_master\_receive\_int function**

Name	Description
Function name	i2c_master_receive_int
Function prototype	i2c_status_type i2c_master_receive_int(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master receives data (interrupt mode). This function is non-blocking, which means that I2C communication has yet to complete after the execution of this function. It is possible to wait for the end of communication by calling the function i2c_wait_end().
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	address: slave address
Input parameter 3	Pdata: array address of data to receive
Input parameter 4	Size: number of data received
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_master_receive_int(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.42 i2c\_slave\_transmit\_int function

The table below describes the function i2c\_slave\_transmit\_int.

**Table 274. 2c\_slave\_transmit\_int function**

Name	Description
Function name	i2c_slave_transmit_int
Function prototype	i2c_status_type i2c_slave_transmit_int(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave sends data (interrupt mode). This function is non-blocking, which means that I2C communication has yet to complete after the execution of this function. It is possible to wait for the end of communication by calling the function i2c_wait_end().
Input parameter 1	hi2c: i2c_handle_type pointer . Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	Pdata: array address of data to send
Input parameter 3	size: number of data transmit
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_slave_transmit_int(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.43 i2c\_slave\_receive\_int function

The table below describes the function i2c\_slave\_receive\_int.

**Table 275. i2c\_slave\_receive\_int function**

Name	Description
Function name	i2c_slave_receive_int
Function prototype	i2c_status_type i2c_slave_receive_int(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave receives data (interrupt mode). This function is non-blocking, which means that I2C communication has yet to complete after the execution of this function. It is possible to wait for the end of communication by calling the function i2c_wait_end().
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	Pdata: array address of data to receive
Input parameter 3	Size: number of data to receive
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_slave_receive_int(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.44 i2c\_master\_transmit\_dma function

The table below describes the function i2c\_master\_transmit\_dma.

**Table 276. i2c\_master\_transmit\_dma function**

Name	Description
Function name	i2c_master_transmit_dma
Function prototype	i2c_status_type i2c_master_transmit_dma(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master sends data (DMA mode). This function is non-blocking, which means that I2C communication has yet to complete after the execution of this function. It is possible to wait for the end of communication by calling the function i2c_wait_end().
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	Address: slave address
Input parameter 3	Pdata: array address of data to send
Input parameter 4	Size: number of data to send
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_master_transmit_dma(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.45 i2c\_master\_receive\_dma function

The table below describes the function i2c\_master\_receive\_dma.

**Table 277. i2c\_master\_receive\_dma function**

Name	Description
Function name	i2c_master_receive_dma
Function prototype	i2c_status_type i2c_master_receive_dma(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Master receives data (DMA mode). This function is non-blocking, which means that I2C communication has yet to complete after the execution of this function. It is possible to wait for the end of communication by calling the function i2c_wait_end().
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	address: slave address
Input parameter 3	Pdata: array address of data to receive
Input parameter 4	Size: number of data to receive
Input parameter 5	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_master_receive_dma(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.46 i2c\_slave\_transmit\_dma function

The table below describes the function i2c\_slave\_transmit\_dma.

**Table 278. i2c\_slave\_transmit\_dma function**

Name	Description
Function name	i2c_slave_transmit_dma
Function prototype	i2c_status_type i2c_slave_transmit_dma(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave sends data (DMA mode). This function is non-blocking, which means that I2C communication has yet to complete after the execution of this function. It is possible to wait for the end of communication by calling the function i2c_wait_end().
Input parameter 1	hi2c: i2c_handle_type pointer. Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	Pdata: array address of data to send
Input parameter 3	Size: number of data to send
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code. Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_slave_transmit_dma(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.47 i2c\_slave\_receive\_dma function

The table below describes the function i2c\_slave\_receive\_dma.

Table 279. i2c\_slave\_receive\_dma function

Name	Description
Function name	i2c_slave_receive_dma
Function prototype	i2c_status_type i2c_slave_receive_dma(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Slave receives data (DMA mode). This function is non-blocking, which means that I2C communication has yet to complete after the execution of this function. It is possible to wait for the end of communication by calling the function i2c_wait_end().
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	pdata: array address of data to receive
Input parameter 3	Size: number of data to receive
Input parameter 4	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

**Example:**

```
i2c_slave_receive_dma(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

## 5.12.48 i2c\_memory\_write function

The table below describes the function i2c\_memory\_write.

**Table 280. i2c\_memory\_write function**

Name	Description
Function name	i2c_memory_write
Function prototype	i2c_status_type i2c_memory_write(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Write EEPROM (polling mode). This function is in blocking mode, which means that the I2C transfer comes to an end after the completion of the function.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the "mem_address_width" description below for details.
Input parameter 3	Address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	Pdata: array address of data to send
Input parameter 6	Size: number of data to send
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

### mem\_address\_width

EEPROM memory address width.

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit

### Example:

```
i2c_memory_write(&hi2c, 0xA0, 0x05, tx_buf, 8, 0xFFFFFFFF);
```

## 5.12.49 i2c\_memory\_write\_int function

The table below describes the function i2c\_memory\_write\_int.

**Table 281. i2c\_memory\_write\_int function**

Name	Description
Function name	i2c_memory_write_int
Function prototype	i2c_status_type i2c_memory_write_int(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Write EEPROM (interrupt mode). This function is non-blocking, which means that the I2C communication has yet to complete after the completion of the function. It is possible to wait for the end of communication by calling the function i2c_wait_end().
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the "mem_address_width" description below for details.
Input parameter 3	Address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to send
Input parameter 6	Size: number of data to send
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

### mem\_address\_width

EEPROM memory address width.

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit

### Example:

```
i2c_memory_write_int(&hi2c, 0xA0, 0x05, tx_buf, 8, 0xFFFFFFFF);
```

## 5.12.50 i2c\_memory\_write\_dma function

The table below describes the function i2c\_memory\_write\_dma.

**Table 282. i2c\_memory\_write\_dma function**

Name	Description
Function name	i2c_memory_write_dma
Function prototype	i2c_status_type i2c_memory_write_dma(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Write EEPROM (DMA mode). This function is non-blocking, which means that the I2C communication has yet to complete after the completion of the function. It is possible to wait for the end of communication by calling the function i2c_wait_end().
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the "mem_address_width" description below for details.
Input parameter 3	Address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	pdata: array address of data to send
Input parameter 6	size: number of data to send
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

### mem\_address\_width

EEPROM memory address width.

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit

### Example:

```
i2c_memory_write_dma(&hi2c, 0xA0, 0x05, tx_buf, 8, 0xFFFFFFFF);
```

## 5.12.51 i2c\_memory\_read function

The table below describes the function i2c\_memory\_read.

**Table 283. i2c\_memory\_read function**

Name	Description
Function name	i2c_memory_read
Function prototype	i2c_status_type i2c_memory_read(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Read EEPROM (polling mode). This function is in blocking mode, which means that the I2C transfer comes to an end after the completion of the function.
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the "mem_address_width" description below for details.
Input parameter 3	Address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	Pdata: array address of data to read
Input parameter 6	Size: number of data to read
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

### mem\_address\_width

EEPROM memory address width.

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit

### Example:

```
i2c_memory_read(&hi2c, 0xA0, 0x05, rx_buf, 8, 0xFFFFFFFF);
```

## 5.12.52 i2c\_memory\_read\_int function

The table below describes the function i2c\_memory\_read\_int.

**Table 284. i2c\_memory\_read\_int function**

Name	Description
Function name	i2c_memory_read_int
Function prototype	i2c_status_type i2c_memory_read_int(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Read EEPROM (interrupt mode). This function is non-blocking, which means that the I2C communication has yet to complete after the completion of the function. It is possible to wait for the end of communication by calling the function
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the "mem_address_width" description below for details.
Input parameter 3	Address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	Pdata: array address of data to read
Input parameter 6	Size: number of data to read
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

### mem\_address\_width

EEPROM memory address width.

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit

### Example:

```
i2c_memory_read_int(&hi2c, 0xA0, 0x05, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.53 i2c\_memory\_read\_dma function

The table below describes the function i2c\_memory\_read\_dma.

**Table 285. i2c\_memory\_read\_dma function**

Name	Description
Function name	i2c_memory_read_dma
Function prototype	i2c_status_type i2c_memory_read_dma(i2c_handle_type* hi2c, i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);
Function description	Read EEPROM (DMA mode). This function is non-blocking, which means that the I2C communication has yet to complete after the completion of the function. It is possible to wait for the end of communication by calling the function
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Input parameter 2	mem_address_width: EEPROM memory address width Refer to the "mem_address_width" description below for details.
Input parameter 3	Address: EEPROM address
Input parameter 4	mem_address: EEPROM data memory address
Input parameter 5	Pdata: array address of data to read
Input parameter 6	Size: number of data to read
Input parameter 7	timeout: wait timeout
Output parameter	NA
Return value	i2c_status_type: error code Refer to <a href="#">section 5.12.32</a> for details.
Required preconditions	NA
Called functions	NA

#### mem\_address\_width

EEPROM memory address width.

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit

#### Example:

```
i2c_memory_read_dma(&hi2c, 0xA0, 0x05, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.54 i2c\_evt\_irq\_handler function

The table below describes the function i2c\_evt\_irq\_handler.

**Table 286. i2c\_evt\_irq\_handler function**

Name	Description
Function name	i2c_evt_irq_handler
Function prototype	void i2c_evt_irq_handler(i2c_handle_type* hi2c);
Function description	Event interrupt function for handling I2C event interrupts
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
void I2C1_EVT_IRQHandler(void)
{
    i2c_evt_irq_handler(&hi2c);
}
```

### 5.12.55 i2c\_err\_irq\_handler function

The table below describes the function i2c\_err\_irq\_handler.

**Table 287. i2c\_err\_irq\_handler function**

Name	Description
Function name	i2c_err_irq_handler
Function prototype	void i2c_err_irq_handler(i2c_handle_type* hi2c);
Function description	Error interrupt function for handling I2C error interrupts
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
void I2C1_ERR_IRQHandler(void)
{
    i2c_err_irq_handler(&hi2c);
}
```

## 5.12.56 i2c\_dma\_tx\_irq\_handler function

The table below describes the function i2c\_dma\_tx\_irq\_handler.

**Table 288. i2c\_dma\_tx\_irq\_handler function**

Name	Description
Function name	i2c_dma_tx_irq_handler
Function prototype	void i2c_dma_tx_irq_handler(i2c_handle_type* hi2c);
Function description	DMA transmit interrupt function for handling DMA transmit interrupts
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
void DMA1_Channel6_IRQHandler(void)
{
    i2c_dma_rx_irq_handler(&hi2c);
}
```

## 5.12.57 i2c\_dma\_rx\_irq\_handler function

The table below describes the function i2c\_dma\_rx\_irq\_handler.

**Table 289. i2c\_dma\_rx\_irq\_handler function**

Name	Description
Function name	i2c_dma_rx_irq_handler
Function prototype	void i2c_dma_rx_irq_handler(i2c_handle_type* hi2c);
Function description	DMA receive interrupt function for handling DMA receive interrupts
Input parameter 1	hi2c: i2c_handle_type pointer Refer to <a href="#">i2c_handle_type</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
void DMA1_Channel7_IRQHandler(void)
{
    i2c_dma_tx_irq_handler(&hi2c);
}
```

## 5.13 Nested vectored interrupt controller (NVIC)

The NVIC register structure NVIC\_Type is defined in the “core\_cm4.h”:

```
/*
 * @brief Structure type to access the Nested Vectored Interrupt Controller (NVIC).
 */
typedef struct
{
    .....
} NVIC_Type;
```

The table below gives a list of the NVIC registers.

**Table 290. Summary of NVIC registers**

Register	Description
isr	Interrupt enable set register
icer	Interrupt enable clear register
ispr	Interrupt suspend set register
icpr	Interrupt suspend clear register
iabr	Interrupt activate bit register
ip	Interrupt priority register
stir	Software trigger interrupt register

The table below gives a list of NVIC library functions.

**Table 291. Summary of NVIC library functions**

Function name	Description
nvic_system_reset	System software reset
nvic_irq_enable	NVIC interrupt enable and priority enable
nvic_irq_disable	NVIC interrupt disable
nvic_priority_group_config	NVIC interrupt priority grouping configuration
nvic_vector_table_set	NVIC interrupt vector table base address and offset address configuration
nvic_lowpower_mode_config	NVIC low-power mode configuration

### 5.13.1 nvic\_system\_reset function

The table below describes the function nvic\_system\_reset.

**Table 292. nvic\_system\_reset function**

Name	Description
Function name	nvic_system_reset
Function prototype	void nvic_system_reset(void)
Function description	System software reset
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_SystemReset()

**Example:**

```
/* system reset */
nvic_system_reset();
```

### 5.13.2 nvic\_irq\_enable function

The table below describes the function nvic\_irq\_enable.

**Table 293. nvic\_irq\_enable function**

Name	Description
Function name	nvic_irq_enable
Function prototype	void nvic_irq_enable(IRQn_Type irqn, uint32_t preempt_priority, uint32_t sub_priority)
Function description	NVIC interrupt enable and priority configuration
Input parameter 1	Irqn: select interrupt vector Refer to the “irqn” descriptions below for details.
Input parameter 2	preempt_priority: set preemption priority This parameter cannot be greater than the highest preemption priority defined in the NVIC_PRIORITY_GROUP_x
Input parameter 3	sub_priority: set response priority This parameter cannot be greater than the highest response priority defined in the NVIC_PRIORITY_GROUP_x
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_SetPriority() NVIC_EnableIRQ()

**irqn**

irqn is used to select interrupt vectors, including:

WWDT\_IRQn: Window timer interrupt

PVM\_IRQn: PVM interrupt linked to EXINT

.....

TMR15\_TRG\_HALL\_IRQn: TMR15 trigger and HALL interrupt

TMR15\_IRQn: TMR15 channel interrupt

**Example:**

```
/* enable nvic irq */
nvic_irq_enable(ADC1_2_IRQn, 0, 0);
```

### 5.13.3 nvic\_irq\_disable function

The table below describes the function nvic\_irq\_disable.

**Table 294. nvic\_irq\_disable function**

Name	Description
Function name	nvic_irq_disable
Function prototype	void nvic_irq_disable(IRQn_Type irqn)
Function description	NVIC interrupt enable
Input parameter	Irqn: select interrupt vector. Refer to "irqn" description for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_DisableIRQ()

**Example:**

```
/* disable nvic irq */
nvic_irq_disable(ADC1_CMP_IRQn);
```

### 5.13.4 nvic\_priority\_group\_config function

The table below describes the function nvic\_priority\_group\_config.

**Table 295. nvic\_priority\_group\_config function**

Name	Description
Function name	nvic_priority_group_config
Function prototype	void nvic_priority_group_config(nvic_priority_group_type priority_group)
Function description	NVIC interrupt priority grouping configuration
Input parameter	priority_group: select interrupt priority group This parameter can be any enumerated value in the nvic_priority_group_type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NVIC_SetPriorityGrouping()

#### priority\_group

priority\_group is used to select priority group, including:

NVIC\_PRIORITY\_GROUP\_0:

Priority group 0 (0 bit for preemption priority, and 4 bits for response priority)

NVIC\_PRIORITY\_GROUP\_1:

Priority group 1 (1 bit for preemption priority, and 3 bits for response priority)

NVIC\_PRIORITY\_GROUP\_2:

Priority group 2 (2 bits for preemption priority, and 2 bits for response priority)

NVIC\_PRIORITY\_GROUP\_3:

Priority group 3 (3 bits for preemption priority, and 1 bit for response priority)

NVIC\_PRIORITY\_GROUP\_4:

Priority group 4 (4 bits for preemption priority, and 0 bit for response priority)

**Example:**

```
/* config nvic priority group */
nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
```

### 5.13.5 nvic\_vector\_table\_set function

The table below describes the function nvic\_vector\_table\_set.

**Table 296. nvic\_vector\_table\_set function**

Name	Description
Function name	nvic_vector_table_set
Function prototype	void nvic_vector_table_set(uint32_t base, uint32_t offset)
Function description	Set NVIC interrupt vector table base address and offset address
Input parameter 1	Base: base address of interrupt vector table The base address can be set in RAM or FLASH
Input parameter 2	Offset: offset address of interrupt vector table This parameter defines the start address of interrupt vector table, so it must be set to a multiple of 0x20.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### base

base is used to select base address of interrupt vector table, including:

NVIC\_VECTTAB\_RAM:      Interrupt vector table base address is located in RAM

NVIC\_VECTTAB\_FLASH:     Interrupt vector table base address is located in FLASH

**Example:**

```
/* config vector table offset */
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x4000);
```

## 5.13.6 nvic\_lowpower\_mode\_config function

The table below describes the function nvic\_lowpower\_mode\_config.

**Table 297. nvic\_lowpower\_mode\_config function**

Name	Description
Function name	nvic_lowpower_mode_config
Function prototype	void nvic_lowpower_mode_config(nvic_lowpower_mode_type lp_mode, confirm_state new_state)
Function description	Configure NVIC low-power mode
Input parameter 1	lp_mode: select low-power modes This parameter can be any enumerated value in the nvic_lowpower_mode_type.
Input parameter 2	new_state: indicates the pre-configured status of battery powered domain This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### lp\_mode

lp\_mode is used to select low-power modes, including:

NVIC\_LP\_SEVONPEND:

Send wakeup event upon interrupt suspend (this option is usually used in conjunction with WFE)

NVIC\_LP\_SLEEPDEEP:

Deepsleep mode control bit (enable or disable core clock))

NVIC\_LP\_SLEEPONEXIT: Sleep mode entry when system leaves the lowest-priority interrupt

### Example:

```
/* enable sleep-on-exit feature */  
nvic_lowpower_mode_config(NVIC_LP_SLEEPONEXIT, TRUE);
```

## 5.14 Power controller (PWC)

The PWC register structure pwc\_type is defined in the “at32f403\_pwc.h”:

```
/*
 * @brief type define pwc register all
 */
typedef struct
{
    .....
} pwc_type;
```

The table below gives a list of the PWC registers.

**Table 298. Summary of PWC registers**

Register	Description
ctrl	Power control register
ctrlsts	Power control/status register

The table below gives a list of PWC library functions.

**Table 299. Summary of PWC library functions**

Function name	Description
pwc_reset	Reset PWC registers to their reset values.
pwc_batteryPoweredDomainAccess	Enable battery powered domain access
pwc_pvm_level_select	Select PVM threshold
pwc_powerVoltageMonitorEnable	Enable Voltage monitor
pwc_wakeupPinEnable	Enable standby-mode wakeup pin
pwc_flagClear	Clear flag
pwc_flagGet	Get flag status
pwc_sleepModeEnter	Enter Sleep mode
pwc_deepSleepModeEnter	Enter Deepsleep mode
pwc_voltageRegulateSet	Select voltage regulator status in Deepsleep mode
pwc_standbyModeEnter	Enter Standby mode

## 5.14.1 pwc\_reset function

The table below describes the function pwc\_reset.

**Table 300. pwc\_reset function**

Name	Description
Function name	pwc_reset
Function prototype	void pwc_reset(void)
Function description	Reset all PWC registers to their reset values.
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset()

**Example:**

```
/* deinitialize pwc */
pwc_reset();
```

## 5.14.2 pwc\_batteryPoweredDomainAccess function

The table below describes the function pwc\_batteryPoweredDomainAccess.

**Table 301. pwc\_batteryPoweredDomainAccess function**

Name	Description
Function name	pwc_batteryPoweredDomainAccess
Function prototype	void pwc_batteryPoweredDomainAccess(confirm_state new_state)
Function description	Battery powered domain access enable
Input parameter	new_state: indicates the pre-configured status of battery powered domain This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable the battery-powered domain write operations */
pwc_batteryPoweredDomainAccess(TRUE);
```

*Note: Access to battery powered domain (such as, RTC) is allowed only after enabling it through this function.*

### 5.14.3 pwc\_pvm\_level\_select function

The table below describes the function pwc\_pvm\_level\_select.

**Table 302. pwc\_pvm\_level\_select function**

Name	Description
Function name	pwc_pvm_level_select
Function prototype	void pwc_pvm_level_select(pwc_pvm_voltage_type pvm_voltage)
Function description	Select PVM threshold
Input parameter	pvm_voltage: indicates the selected PVM threshold This parameter can be any enumerated value in the pwc_pvm_voltage_type.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### pvm\_voltage

pvm\_voltage is used to select PVM threshold, including:

PWC\_PVM\_VOLTAGE\_2V3: PVM threshold is 2.3V  
 PWC\_PVM\_VOLTAGE\_2V4: PVM threshold is 2.4V  
 PWC\_PVM\_VOLTAGE\_2V5: PVM threshold is 2.5V  
 PWC\_PVM\_VOLTAGE\_2V6: PVM threshold is 2.6V  
 PWC\_PVM\_VOLTAGE\_2V7: PVM threshold is 2.7V  
 PWC\_PVM\_VOLTAGE\_2V8: PVM threshold is 2.8V  
 PWC\_PVM\_VOLTAGE\_2V9: PVM threshold is 2.9V

#### Example:

```
/* set the threshold voltage to 2.9v */
pwc_pvm_level_select(PWC_PVM_VOLTAGE_2V9);
```

### 5.14.4 pwc\_power\_voltage\_monitor\_enable function

The table below describes the function pwc\_power\_voltage\_monitor\_enable.

**Table 303. pwc\_power\_voltage\_monitor\_enable function**

Name	Description
Function name	pwc_power_voltage_monitor_enable
Function prototype	void pwc_power_voltage_monitor_enable(confirm_state new_state)
Function description	Enable power voltage monitor (PVM)
Input parameter	new_state: indicates the pre-configured status of PVM This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### Example:

```
/* enable power voltage monitor */
pwc_power_voltage_monitor_enable(TRUE);
```

## 5.14.5 pwc\_wakeup\_pin\_enable function

The table below describes the function pwc\_wakeup\_pin\_enable.

**Table 304. pwc\_wakeup\_pin\_enable function**

Name	Description
Function name	pwc_wakeup_pin_enable
Function prototype	void pwc_wakeup_pin_enable(uint32_t pin_num, confirm_state new_state)
Function description	Enable Standby wakeup pin
Input parameter 1	pin_num: select a standby wakeup pins This parameter can be any pin that is capable of waking up from Standby mode.
Input parameter 2	new_state: indicates the pre-configured status of Standby wakeup pins This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### pin\_num

pin\_num is used to select Standby-mode wakeup pin, including:

PWC\_WAKEUP\_PIN\_1: Standby wakeup pin 1 (corresponding GPIO is PA0)

### Example:

```
/* enable wakeup pin - pa0 */
pwc_wakeup_pin_enable(PWC_WAKEUP_PIN_1, TRUE);
```

## 5.14.6 pwc\_flag\_clear function

The table below describes the function pwc\_flag\_clear.

**Table 305. pwc\_flag\_clear function**

Name	Description
Function name	pwc_flag_clear
Function prototype	void pwc_flag_clear(uint32_t pwc_flag)
Function description	Clear flag
Input parameter	pwc_flag: select flag to clear Refer to the "pwc_flag" description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### pwc\_flag

pwc\_flag is used to select flags, including:

PWC\_WAKEUP\_FLAG: Standby wakeup event

PWC\_STANDBY\_FLAG: Standby mode entry

PWC\_PVM\_OUTPUT\_FLAG: PVM output (this parameter cannot be cleared by software)

### Example:

```
/* wakeup event flag clear */
pwc_flag_clear(PWC_WAKEUP_FLAG);
```

## 5.14.7 pwc\_flag\_get function

The table below describes the function pwc\_flag\_get.

**Table 306. pwc\_flag\_get function**

Name	Description
Function name	pwc_flag_get
Function prototype	flag_status pwc_flag_get(uint32_t pwc_flag)
Function description	Get flag status
Input parameter	pwc_flag: select a flag. Refer to <a href="#">pwc_flag</a> for details.
Output parameter	NA
Return value	flag_status: indicates flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

**Example:**

```
/* check if wakeup event flag is set */
if(pwc_flag_get(PWC_WAKEUP_FLAG) != RESET)
```

## 5.14.8 pwc\_sleep\_mode\_enter function

The table below describes the function pwc\_sleep\_mode\_enter.

**Table 307. pwc\_sleep\_mode\_enter function**

Name	Description
Function name	pwc_sleep_mode_enter
Function prototype	void pwc_sleep_mode_enter(pwc_sleep_enter_type pwc_sleep_enter)
Function description	Enter Sleep mode
Input parameter	pwc_sleep_enter: select a command to enter Sleep mode This parameter can be any enumerated value in the pwc_sleep_enter_type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**pwc\_sleep\_enter**

pwc\_sleep\_enter is used to select a command to enter Sleep mode, including:

PWC\_SLEEP\_ENTER\_WFI: Enter Sleep mode by WFI

PWC\_SLEEP\_ENTER\_WFE: Enter Sleep mode by WFE

**Example:**

```
/* enter sleep mode */
pwc_sleep_mode_enter(PWC_SLEEP_ENTER_WFI);
```

## 5.14.9 pwc\_deep\_sleep\_mode\_enter function

The table below describes the function pwc\_deep\_sleep\_mode\_enter.

**Table 308. pwc\_deep\_sleep\_mode\_enter function**

Name	Description
Function name	pwc_deep_sleep_mode_enter
Function prototype	void pwc_deep_sleep_mode_enter(pwc_deep_sleep_enter_type pwc_deep_sleep_enter)
Function description	Enter Deepsleep mode
Input parameter	pwc_deep_sleep_enter: select a command to enter Deepsleep mode This parameter can be any enumerated value in the pwc_deep_sleep_enter_type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### **pwc\_deep\_sleep\_enter**

pwc\_deep\_sleep\_enter is used to select a command to enter Deepsleep mode, including:

PWC\_DEEP\_SLEEP\_ENTER\_WFI: Enter Deepsleep mode by WFI

PWC\_DEEP\_SLEEP\_ENTER\_WFE: Enter Deepsleep mode by WFE

#### **Example:**

```
/* enter deep sleep mode */  
pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);
```

### 5.14.10 pwc\_standby\_mode\_enter function

The table below describes the function pwc\_standby\_mode\_enter

**Table 309. pwc\_standby\_mode\_enter function**

Name	Description
Function name	pwc_standby_mode_enter
Function prototype	void pwc_standby_mode_enter(void)
Function description	Enter Standby mode
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enter standby mode */  
pwc_standby_mode_enter();
```

## 5.15 Real-time clock (RTC)

The RTC register structure ertc\_type is defined in the “at32f403\_RTC.h”:

```
/*
 * @brief type define rtc register all
 */
typedef struct
{
    } rtc_type;
```

The table below gives a list of the ERTC registers.

**Table 310. Summary of RTC registers**

Register	Description
ctrlh	RTC control register high
ctrll	RTC control register low
divh	RTC divider register high
divl	RTC divider register low
divcnth	RTC divider counter register high
divcntl	RTC divider counter register low
cnth	RTC counter value register high
cntl	RTC counter value register low
tah	RTC alarm register high
tal	RTC alarm register low

The table below gives a list of RTC library functions.

**Table 311. Summary of RTC library functions**

Function name	Description
rtc_counter_set	Set RTC counter value
rtc_counter_get	Get RTC counter value
rtc_divider_set	Set RTC divider
rtc_divider_get	Get RTC divider
rtc_alarm_set	Set RTC alarm
rtc_interrupt_enable	Enable RTC interrupts
rtc_flag_get	Get RTC flag
rtc_flag_clear	Clear RTC flag
rtc_wait_config_finish	RTC waits for configuration complete
rtc_wait_update_finish	RTC waits for time update complete

### 5.15.1 rtc\_counter\_set function

The table below describes the function rtc\_counter\_set.

**Table 312. rtc\_counter\_set function**

Name	Description
Function name	rtc_counter_set
Function prototype	void rtc_counter_set(uint32_t counter_value);
Function description	Set RTC counter value
Input parameter 1	counter_value: RTC counter value (range 0~0xFFFFFFFF)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
rtc_counter_set(0x00000008);
```

### 5.15.2 rtc\_counter\_get function

The table below describes the function rtc\_counter\_get.

**Table 313. rtc\_counter\_get function**

Name	Description
Function name	rtc_counter_get
Function prototype	uint32_t rtc_counter_get(void);
Function description	Get RTC counter value
Input parameter 1	NA
Output parameter	NA
Return value	uint32_t: the current counter value. The value is usually incremented by 1 per second.
Required preconditions	NA
Called functions	NA

**Example:**

```
value = rtc_counter_get();
```

### 5.15.3 rtc\_divider\_set function

The table below describes the function rtc\_divider\_set.

**Table 314. rtc\_divider\_set function**

Name	Description
Function name	rtc_divider_set
Function prototype	void rtc_divider_set(uint32_t div_value);
Function description	Set RTC divider
Input parameter 1	div_value: RTC divider value (range 0~0x000FFFFF)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
rtc_divider_set(32767);
```

### 5.15.4 rtc\_divider\_get function

The table below describes the function rtc\_divider\_get.

**Table 315. rtc\_divider\_get function**

Name	Description
Function name	rtc_divider_get
Function prototype	uint32_t rtc_divider_get(void);
Function description	Get RTC divider value
Input parameter 1	NA
Output parameter	NA
Return value	uint32_t: the current divider value
Required preconditions	NA
Called functions	NA

**Example:**

```
value = rtc_divider_get();
```

## 5.15.5 rtc\_alarm\_set function

The table below describes the function `rtc_alarm_set`.

**Table 316. `rtc_alarm_set` function**

Name	Description
Function name	<code>rtc_alarm_set</code>
Function prototype	<code>void rtc_alarm_set(uint32_t alarm_value);</code>
Function description	Set RTC alarm
Input parameter 1	alarm_value: RTC alarm value, range 0~0xFFFFFFFF. An alarm event occurs when RTC counter value equals to the alarm value.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
rtc_alarm_set(0x00000006);
```

## 5.15.6 rtc\_interrupt\_enable function

The table below describes the function `rtc_interrupt_enable`.

**Table 317. `rtc_interrupt_enable` function**

Name	Description
Function name	<code>rtc_interrupt_enable</code>
Function prototype	<code>void rtc_interrupt_enable(uint16_t source, confirm_state new_state);</code>
Function description	Enable interrupts
Input parameter 1	Source: interrupt source. Refer to the “source” descriptions below for details.
Input parameter 2	new_state: indicates the status of interrupt This parameter can be TRUE or FALSE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Source**

Interrupt source selection.

`RTC_TS_INT`: Second interrupt

`RTC_TA_INT`: Alarm interrupt

`RTC_OVF_INT`: Counter overflow interrupt

**Example:**

```
rtc_interrupt_enable(RTC_TS_INT, TRUE);
```

## 5.15.7 rtc\_flag\_get function

The table below describes the function `rtc_flag_get`.

**Table 318. `rtc_flag_get` function**

Name	Description
Function name	<code>rtc_flag_get</code>
Function prototype	<code>flag_status rtc_flag_get(uint16_t flag);</code>
Function description	Get flag status
Input parameter 1	Flag: flag selection Refer to the “flag” descriptions below for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET
Required preconditions	NA
Called functions	NA

### flag

This is used for flag selection, including:

- RTC\_TS\_FLAG: Second flag
- RTC\_TA\_FLAG: Alarm flag
- RTC\_OVF\_FLAG: Counter overflow flag
- RTC\_UPDF\_FLAG: Time update flag
- RTC\_CCFG\_FLAG: RTC register configuration complete flag

### Example:

```
rtc_flag_get(RTC_TS_FLAG);
```

## 5.15.8 rtc\_interrupt\_flag\_get

The table below describes the function `rtc_interrupt_flag_get`.

**Table 319. `rtc_interrupt_flag_get` function**

Name	Description
Function name	<code>rtc_interrupt_flag_get</code>
Function prototype	<code>flag_status rtc_interrupt_flag_get(uint16_t flag);</code>
Function description	Get flag status and judge the corresponding interrupt enable bit
Input parameter 1	flag: flag selection. Refer to the “flag” descriptions below for details.
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET
Required preconditions	NA
Called functions	NA

### flag

This is used for flag selection, including:

- RTC\_TS\_FLAG: Second flag
- RTC\_TA\_FLAG: Alarm flag
- RTC\_OVF\_FLAG: Counter overflow flag

### Example

```
rtc_interrupt_flag_get(RTC_TS_FLAG);
```

### 5.15.9 rtc\_flag\_clear function

The table below describes the function rtc\_flag\_clear.

Table 320. rtc\_flag\_clear function

Name	Description
Function name	rtc_flag_clear
Function prototype	void rtc_flag_clear(uint16_t flag);
Function description	Clear flag
Input parameter 1	Flag: flag selection Refer to the “flag” descriptions below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### flag

This is used for flag selection, including:

- RTC\_TS\_FLAG: Second flag  
RTC\_TA\_FLAG: Alarm flag  
RTC\_OVF\_FLAG: Counter overflow flag  
RTC\_UPDF\_FLAG: Time update flag

#### Example:

```
rtc_init_mode_exit();
```

### 5.15.10 rtc\_wait\_config\_finish function

The table below describes the function rtc\_wait\_config\_finish.

Table 321. rtc\_wait\_config\_finish function

Name	Description
Function name	rtc_wait_config_finish
Function prototype	void rtc_wait_config_finish(void);
Function description	RTC waits for the completion of configuration
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### Example:

```
rtc_wait_config_finish();
```

### 5.15.11 rtc\_wait\_update\_finish function

The table below describes the function rtc\_wait\_update\_finish.

**Table 322. rtc\_wait\_update\_finish function**

Name	Description
Function name	rtc_wait_update_finish
Function prototype	void rtc_wait_update_finish(void);
Function description	RTC waits for the completion of time update
Input parameter 1	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
rtc_wait_update_finish();
```

## 5.16 Serial peripheral interface (SPI)/ I<sup>2</sup>S

The SPI register structure spi\_type is defined in the “at32f403\_spi.h”:

```
/*
 * @brief type define spi register all
 */
typedef struct
{
    ...
} spi_type;
```

The table below gives a list of the SPI registers.

**Table 323. Summary of SPI registers**

Register	Description
ctrl1	SPI control register 1
ctrl2	SPI control register 2
sts	SPI status register
dt	SPI data register
cpoly	SPI CRC register
rcrc	SPI RxCRC register
tcrc	SPI TxCRC register
i2sctrl	SPI_I2S configuration register
i2sclkp	SPI_I2S prescaler register

The table below gives a list of SPI library functions.

**Table 324. Summary of SPI library functions**

Function name	Description
spi_i2s_reset	Reset SPI/I <sup>2</sup> S registers to their reset values
spi_default_para_init	Configure the SPI initialization structure with an initial value
spi_init	Initialize SPI
spi_crc_next_transmit	Next data transfer is CRC command
spi_crc_polynomial_set	SPI CRC polynomial configuration
spi_crc_polynomial_get	Get SPI CRC polynomial
spi_crc_enable	Enable SPI CRC
spi_crc_value_get	Get CRC result of SPI receive/transmit
spi_hardware_cs_output_enable	Enable hardware CS output
spi_software_cs_internal_level_set	Set software CS internal level
spi_frame_bit_num_set	Set the number of frame bits
spi_half_duplex_direction_set	Set transfer direction of single-wire bidirectional half-duplex mode
spi_enable	Enable SPI
i2s_default_para_init	Set an initial value for the I <sup>2</sup> S initialization structure
i2s_init	Initialize I <sup>2</sup> S
i2s_enable	Enable I <sup>2</sup> S
spi_i2s_interrupt_enable	Enable SPI/I <sup>2</sup> S interrupts
spi_i2s_dma_transmitter_enable	Enable SPI/I <sup>2</sup> S DMA transmit

spi_i2s_dma_receiver_enable	Enable SPI/I <sup>2</sup> S DMA receive
spi_i2s_data_transmit	SPI/I <sup>2</sup> S transmits data
spi_i2s_data_receive	SPI/I <sup>2</sup> S receives data
spi_i2s_flag_get	Get SPI/I <sup>2</sup> S flags
spi_i2s_flag_clear	Clear SPI/I <sup>2</sup> S flags

### 5.16.1 spi\_i2s\_reset function

The table below describes the function spi\_i2s\_reset.

**Table 325. spi\_i2s\_reset function**

Name	Description
Function name	spi_i2s_reset
Function prototype	void spi_i2s_reset(spi_type *spi_x);
Function description	Reset SPI/I <sup>2</sup> S registers to their reset values.
Input parameter 1	spi_x: select SPI peripherals This parameter can be SPI1, SPI2, SPI3, SPI4
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

**Example:**

```
spi_i2s_reset (SPI1);
```

### 5.16.2 spi\_default\_para\_init function

The table below describes the function spi\_default\_para\_init.

**Table 326. spi\_default\_para\_init function**

Name	Description
Function name	spi_default_para_init
Function prototype	void spi_default_para_init(spi_init_type* spi_init_struct);
Function description	Set an initial value for the SPI initialization structure
Input parameter 1	spi_init_struct: <a href="#">spi_init_type</a> pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of spi_init_type before starting.
Called functions	NA

**Example:**

```
spi_init_type spi_init_struct;  
spi_default_para_init (&spi_init_struct);
```

### 5.16.3 spi\_init function

The table below describes the function spi\_init.

**Table 327. spi\_init function**

Name	Description
Function name	spi_init
Function prototype	void spi_init(spi_type* spi_x, spi_init_type* spi_init_struct);
Function description	Initialize SPI
Input parameter 1	spi_x: select SPI peripherals This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	spi_init_struct: <i>spi_init_type</i> pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of <i>spi_init_type</i> before starting.
Called functions	NA

spi\_init\_type is defined in the at32f403\_spi.h:

typedef struct

```
{
    spi_transmission_mode_type      transmission_mode;
    spi_master_slave_mode_type     master_slave_mode;
    spi_mclk_freq_div_type        mclk_freq_division;
    spi_first_bit_type            first_bit_transmission;
    spi_frame_bit_num_type        frame_bit_num;
    spi_clock_polarity_type       clock_polarity;
    spi_clock_phase_type          clock_phase;
    spi_cs_mode_type              cs_mode_selection;
} spi_init_type;
```

#### spi\_transmission\_mode

SPI transmission mode.

SPI_TRANSMIT_FULL_DUPLEX:	Two-wire unidirectional full-duplex mode
SPI_TRANSMIT_SIMPLEX_RX:	Two-wire unidirectional receive-only mode
SPI_TRANSMIT_HALF_DUPLEX_RX:	Single-wire bidirectional receive-only mode
SPI_TRANSMIT_HALF_DUPLEX_TX:	Single-wire bidirectional transmit-only mode

#### master\_slave\_mode

Master/slave mode selection.

SPI_MODE_SLAVE:	Slave mode
SPI_MODE_MASTER:	Master mode

#### mclk\_freq\_division

Frequency division factor selection.

SPI_MCLK_DIV_2:	Divided by 2
SPI_MCLK_DIV_4:	Divided by 4
SPI_MCLK_DIV_8:	Divided by 8
SPI_MCLK_DIV_16:	Divided by 16
SPI_MCLK_DIV_32:	Divided by 32
SPI_MCLK_DIV_64:	Divided by 64
SPI_MCLK_DIV_128:	Divided by 128

SPI\_MCLK\_DIV\_256: Divided by 256  
SPI\_MCLK\_DIV\_512: Divided by 512  
SPI\_MCLK\_DIV\_1024: Divided by 1024

**first\_bit\_transmission**

SPI MSB-first/LSB-first selection  
SPI\_FIRST\_BIT\_MSB: MSB-first  
SPI\_FIRST\_BIT\_LSB: LSB-first

**frame\_bit\_num**

Set the number of bits in a frame  
SPI\_FRAME\_8BIT: 8-bit data in a frame  
SPI\_FRAME\_16BIT: 16-bit data in a frame

**clock\_polarity**

Select Clock polarity.  
SPI\_CLOCK\_POLARITY\_LOW: Clock output low in idle state  
SPI\_CLOCK\_POLARITY\_HIGH: Clock output high in idle state

**clock\_phase**

Select clock phase.  
SPI\_CLOCK\_PHASE\_1EDGE: Sample on the first clock edge  
SPI\_CLOCK\_PHASE\_2EDGE: Sample on the second clock edge

**cs\_mode\_selection**

Select CS mode.  
SPI\_CS\_HARDWARE\_MODE: Hardware CS mode  
SPI\_CS\_SOFTWARE\_MODE: Software CS mode

**Example:**

```
spi_init_type spi_init_struct;
spi_default_para_init(&spi_init_struct);
spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;
spi_init_struct.master_slave_mode = SPI_MODE_MASTER;
spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;
spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_MSB;
spi_init_struct.frame_bit_num = SPI_FRAME_16BIT;
spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;
spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;
spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;
spi_init(SPI1, &spi_init_struct);
```

## 5.16.4 spi\_crc\_next\_transmit function

The table below describes the function spi\_crc\_next\_transmit.

**Table 328. spi\_crc\_next\_transmit function**

Name	Description
Function name	spi_crc_next_transmit
Function prototype	void spi_crc_next_transmit(spi_type* spi_x);
Function description	The next data to be sent is CRC command
Input parameter 1	spi_x: select SPI peripherals This parameter can be SPI1, SPI2, SPI3, SPI4.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
spi_crc_next_transmit (SPI1);
```

## 5.16.5 spi\_crc\_polynomial\_set function

The table below describes the function spi\_crc\_polynomial\_set.

**Table 329. spi\_crc\_polynomial\_set function**

Name	Description
Function name	spi_crc_polynomial_set
Function prototype	void spi_crc_polynomial_set(spi_type* spi_x, uint16_t crc_poly);
Function description	Set SPI CRC polynomial
Input parameter 1	spi_x: select SPI peripherals This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	crc_poly: CRC polynomial Value is 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/*set spi crc polynomial value */  
spi_crc_polynomial_set (SPI1, 0x07);
```

## 5.16.6 spi\_crc\_polynomial\_get function

The table below describes the function spi\_crc\_polynomial\_get.

**Table 330. spi\_crc\_polynomial\_get function**

Name	Description
Function name	spi_crc_polynomial_get
Function prototype	uint16_t spi_crc_polynomial_get(spi_type* spi_x);
Function description	Get SPI CRC polynomial
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4.
Output parameter	NA
Return value	CRC polynomial Value is 0x0000~0xFFFF
Required preconditions	NA
Called functions	NA

**Example:**

```
/*get spi crc polynomial value */
uint16_t crc_poly;
crc_poly = spi_crc_polynomial_get (SPI1);
```

## 5.16.7 spi\_crc\_enable function

The table below describes the function spi\_crc\_enable.

**Table 331. spi\_crc\_enable function**

Name	Description
Function name	spi_crc_enable
Function prototype	void spi_crc_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI CRC
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* spi crc enable */
spi_crc_enable (SPI1, TRUE);
```

## 5.16.8 spi\_crc\_value\_get function

The table below describes the function spi\_crc\_value\_get.

**Table 332. spi\_crc\_value\_get function**

Name	Description
Function name	spi_crc_value_get
Function prototype	uint16_t spi_crc_value_get(spi_type* spi_x, spi_crc_direction_type crc_direction);
Function description	Get SPI receive/transmit CRC result
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	<i>crc_direction</i> : Select receive/transmit CRC
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### crc\_direction

Select receive/transmit CRC

SPI\_CRC\_RX: Receive CRC

SPI\_CRC\_TX: Transmit CRC

#### Example:

```
/* get spi rx & tx crc enable */
uint16_t spi_rx_crc, spi_tx_crc;
spi_rx_crc = spi_crc_value_get (SPI1, SPI_CRC_RX);
spi_tx_crc = spi_crc_value_get (SPI1, SPI_CRC_TX);
```

## 5.16.9 spi\_hardware\_cs\_output\_enable function

The table below describes the function spi\_hardware\_cs\_output\_enable.

**Table 333. spi\_hardware\_cs\_output\_enable function**

Name	Description
Function name	spi_hardware_cs_output_enable
Function prototype	void spi_hardware_cs_output_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable hardware CS output
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	new_state: Enable or disable This parameter can FALSE or TRUE
Output parameter	NA
Return value	NA
Required preconditions	This setting is applicable to SPI master mode only.
Called functions	NA

#### Example:

```
/* enable the hardware cs output */
spi_hardware_cs_output_enable (SPI1, TRUE);
```

## 5.16.10 spi\_software\_cs\_internal\_level\_set function

The table below describes the function spi\_software\_cs\_internal\_level\_set.

**Table 334. spi\_software\_cs\_internal\_level\_set function**

Name	Description
Function name	spi_software_cs_internal_level_set
Function prototype	void spi_software_cs_internal_level_set(spi_type* spi_x, spi_software_cs_level_type level);
Function description	Set software CS internal level
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	<i>level</i> : set software CS internal level
Output parameter	NA
Return value	NA
Required preconditions	1. This setting is applicable to software CS mode only; 2. In master mode, the “level” value must be “SPI_SWCS_INTERNAL_LEVEL_HIGHT”.
Called functions	NA

### level

Set software CS internal level

SPI\_SWCS\_INTERNAL\_LEVEL\_LOW: Software CS internal low level

SPI\_SWCS\_INTERNAL\_LEVEL\_HIGHT: Software CS internal high level

### Example:

```
/* set the internal level high */  
spi_software_cs_internal_level_set (SPI1, SPI_SWCS_INTERNAL_LEVEL_HIGHT);
```

### 5.16.11 spi\_frame\_bit\_num\_set function

The table below describes the function spi\_frame\_bit\_num\_set.

**Table 335. spi\_frame\_bit\_num\_set function**

Name	Description
Function name	spi_frame_bit_num_set
Function prototype	void spi_frame_bit_num_set(spi_type* spi_x, spi_frame_bit_num_type bit_num);
Function description	Set the number of bits in a frame
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	<i>bit_num</i> : Set the number of bits in a frame
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### bit\_num

Set the number of bits in a frame

SPI\_FRAME\_8BIT: 8-bit data in a frame

SPI\_FRAME\_16BIT: 16-bit data in a frame

#### Example:

```
/* set the data frame bit num as 8 */
spi_frame_bit_num_set (SPI1, SPI_FRAME_8BIT);
```

### 5.16.12 spi\_half\_duplex\_direction\_set function

The table below describes the function spi\_half\_duplex\_direction\_set.

**Table 336. spi\_half\_duplex\_direction\_set function**

Name	Description
Function name	spi_half_duplex_direction_set
Function prototype	void spi_half_duplex_direction_set(spi_type* spi_x, spi_half_duplex_direction_type direction);
Function description	Set the transfer direction of single-wire bidirectional half-duplex mode
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	<i>direction</i> : transfer direction
Output parameter	NA
Return value	NA
Required preconditions	This setting is applicable to the single-wire bidirectional half-duplex mode only.
Called functions	NA

#### direction

Transfer direction

SPI\_HALF\_DUPLEX\_DIRECTION\_RX: Receive

SPI\_HALF\_DUPLEX\_DIRECTION\_TX: Transmit

#### Example:

```
/* set the data transmission direction as transmit */
spi_half_duplex_direction_set (SPI1, SPI_HALF_DUPLEX_DIRECTION_TX);
```

### 5.16.13 spi\_enable function

The table below describes the function spi\_enable.

**Table 337. spi\_enable function**

Name	Description
Function name	spi_enable
Function prototype	void spi_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable spi */
spi_enable (SPI1, TRUE);
```

### 5.16.14 i2s\_default\_para\_init function

The table below describes the function i2s\_default\_para\_init.

**Table 338. i2s\_default\_para\_init function**

Name	Description
Function name	i2s_default_para_init
Function prototype	void i2s_default_para_init(i2s_init_type* i2s_init_struct);
Function description	Set an initial value for the I <sup>2</sup> S initialization structure
Input parameter 1	i2s_init_struct: <i>spi_i2s_flag</i> pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of i2s_init_type before starting.
Called functions	NA

**Example:**

```
i2s_init_type i2s_init_struct;
i2s_default_para_init (&i2s_init_struct);
```

## 5.16.15 i2s\_init function

The table below describes the function i2s\_init.

**Table 339. i2s\_init function**

Name	Description
Function name	i2s_init
Function prototype	void i2s_init(spi_type* spi_x, i2s_init_type* i2s_init_struct);
Function description	Initialize I <sup>2</sup> S
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4
Input parameter 2	i2s_init_struct: <i>spi_i2s_flag</i> pointer
Output parameter	NA
Return value	NA
Required preconditions	It is necessary to define a variable of i2s_init_type before starting.
Called functions	NA

i2s\_init\_type is defined in the at32f403\_spi.h:

typedef struct

{

i2s_operation_mode_type	operation_mode;
i2s_audio_protocol_type	audio_protocol;
i2s_audio_sampling_freq_type	audio_sampling_freq;
i2s_data_channel_format_type	data_channel_format;
i2s_clock_polarity_type	clock_polarity;
confirm_state	mclk_output_enable;

} i2s\_init\_type;

### **operation\_mode**

I<sup>2</sup>S transfer mode

I2S_MODE_SLAVE_TX:	I2S slave transmit
I2S_MODE_SLAVE_RX:	I2S slave receive
I2S_MODE_MASTER_TX:	I2S master transmit
I2S_MODE_MASTER_RX:	I2S master receive

### **audio\_protocol**

I<sup>2</sup>S audio protocol standards

I2S_AUDIO_PROTOCOL_PHILLIPS:	Phillips
I2S_AUDIO_PROTOCOL_MSB:	MSB aligned (left-aligned)
I2S_AUDIO_PROTOCOL_LSB:	LSB aligned (right-aligned)
I2S_AUDIO_PROTOCOL_PCM_SHORT:	PCM short frame synchronization
I2S_AUDIO_PROTOCOL_PCM_LONG:	PCM long frame synchronization

### **audio\_sampling\_freq**

I<sup>2</sup>S audio sampling frequency.

I2S\_AUDIO\_FREQUENCY\_DEFAULT:

Kept at its reset value (sampling frequency changes with SCLK)

I2S_AUDIO_FREQUENCY_8K:	I2S sampling frequency 8K
I2S_AUDIO_FREQUENCY_11_025K:	I2S sampling frequency 11.025K
I2S_AUDIO_FREQUENCY_16K:	I2S sampling frequency 16K
I2S_AUDIO_FREQUENCY_22_05K:	I2S sampling frequency 22.05K
I2S_AUDIO_FREQUENCY_32K:	I2S sampling frequency 32K

I2S\_AUDIO\_FREQUENCY\_44\_1K: I2S sampling frequency 44.1K  
 I2S\_AUDIO\_FREQUENCY\_48K: I2S sampling frequency 48K  
 I2S\_AUDIO\_FREQUENCY\_96K: I2S sampling frequency 96K  
 I2S\_AUDIO\_FREQUENCY\_192K: I2S sampling frequency 192K

#### **data\_channel\_format**

I<sup>2</sup>S data/channel bits format

I2S\_DATA\_16BIT\_CHANNEL\_16BIT: 16-bit data, 16-bit channel  
 I2S\_DATA\_16BIT\_CHANNEL\_32BIT: 16-bit data, 32-bit channel  
 I2S\_DATA\_24BIT\_CHANNEL\_32BIT: 24-bit data, 32-bit channel  
 I2S\_DATA\_32BIT\_CHANNEL\_32BIT: 32-bit data, 32-bit channel

#### **clock\_polarity**

I<sup>2</sup>S clock polarity

I2S\_CLOCK\_POLARITY\_LOW: Clock output low in idle state  
 I2S\_CLOCK\_POLARITY\_HIGH: Clock output high in idle state

#### **mclk\_output\_enable**

Enable mclk clock output

This parameter can be FALSE or TRUE.

#### **Example:**

```
i2s_init_type i2s_init_struct;
i2s_default_para_init(&i2s_init_struct);
i2s_init_struct.audio_protocol = I2S_AUDIO_PROTOCOL_PHILLIPS;
i2s_init_struct.data_channel_format = I2S_DATA_16BIT_CHANNEL_32BIT;
i2s_init_struct.mclk_output_enable = FALSE;
i2s_init_struct.audio_sampling_freq = I2S_AUDIO_FREQUENCY_48K;
i2s_init_struct.clock_polarity = I2S_CLOCK_POLARITY_LOW;
i2s_init_struct.operation_mode = I2S_MODE_MASTER_TX;
i2s_init(SPI2, &i2s_init_struct);
```

### 5.16.16 i2s\_enable function

The table below describes the function i2s\_enable.

**Table 340. i2s\_enable function**

Name	Description
Function name	i2s_enable
Function prototype	void i2s_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable I <sup>2</sup> S
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4.
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### **Example:**

```
/* enable i2s*/
i2s_enable (SPI1, TRUE);
```

## 5.16.17 spi\_i2s\_interrupt\_enable function

The table below describes the function spi\_i2s\_interrupt\_enable.

**Table 341. spi\_i2s\_interrupt\_enable function**

Name	Description
Function name	spi_i2s_interrupt_enable
Function prototype	void spi_i2s_interrupt_enable(spi_type* spi_x, uint32_t spi_i2s_int, confirm_state new_state);
Function description	Enable SPI/I <sup>2</sup> S interrupts
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4
Input parameter 2	<i>spi_i2s_int</i> : select SPI interrupts
Input parameter 3	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### spi\_i2s\_int

Select SPI/I<sup>2</sup>S interrupts.

SPI\_I2S\_ERROR\_INT: SPI/I<sup>2</sup>S error interrupts (including CRC error, overflow error, underflow error and mode error)

SPI\_I2S\_RDBF\_INT: Receive data buffer full

SPI\_I2S\_TDBE\_INT: Transmit data buffer empty

### Example:

```
/* enable the specified spi/i2s interrupts */  
spi_i2s_interrupt_enable (SPI1, SPI_I2S_ERROR_INT);  
spi_i2s_interrupt_enable (SPI1, SPI_I2S_RDBF_INT);  
spi_i2s_interrupt_enable (SPI1, SPI_I2S_TDBE_INT);
```

## 5.16.18 spi\_i2s\_dma\_transmitter\_enable function

The table below describes the function spi\_i2s\_dma\_transmitter\_enable.

**Table 342. spi\_i2s\_dma\_transmitter\_enable function**

Name	Description
Function name	spi_i2s_dma_transmitter_enable
Function prototype	void spi_i2s_dma_transmitter_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI/I <sup>2</sup> S DMA transmitter
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable spi transmitter dma */
spi_i2s_dma_transmitter_enable (SPI1, TRUE);
```

## 5.16.19 spi\_i2s\_dma\_receiver\_enable function

The table below describes the function spi\_i2s\_dma\_receiver\_enable.

**Table 343. spi\_i2s\_dma\_receiver\_enable function**

Name	Description
Function name	spi_i2s_dma_receiver_enable
Function prototype	void spi_i2s_dma_receiver_enable(spi_type* spi_x, confirm_state new_state);
Function description	Enable SPI/I <sup>2</sup> S DMA receiver
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4
Input parameter 2	new_state: Enable or disable This parameter can be FALSE or TRUE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable spi dma transmitter */
spi_i2s_dma_transmitter_enable (SPI1, TRUE);
```

## 5.16.20 spi\_i2s\_data\_transmit function

The table below describes the function spi\_i2s\_data\_transmit.

**Table 344. spi\_i2s\_data\_transmit function**

Name	Description
Function name	spi_i2s_data_transmit
Function prototype	void spi_i2s_data_transmit(spi_type* spi_x, uint16_t tx_data);
Function description	SPI/I <sup>2</sup> S sends data
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4
Input parameter 2	tx_data: data to send Value range (for 8-bit bit in a frame): 0x00~0xFF Value range (for 16-bit in a frame): 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* spi data transmit */
uint16_t tx_data = 0x6666;
spi_i2s_data_transmit (SPI1, tx_data);
```

## 5.16.21 spi\_i2s\_data\_receive function

The table below describes the function spi\_i2s\_data\_receive.

**Table 345. spi\_i2s\_data\_receive function**

Name	Description
Function name	spi_i2s_data_receive
Function prototype	uint16_t spi_i2s_data_receive(spi_type* spi_x);
Function description	SPI/I <sup>2</sup> S receives data
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4
Output parameter	rx_data: data to receive Value range (for 8-bit bit in a frame): 0x00~0xFF Value range (for 16-bit in a frame): 0x0000~0xFFFF
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* spi data receive */
uint16_t rx_data = 0;
rx_data = spi_i2s_data_receive (SPI1);
```

## 5.16.22 spi\_i2s\_flag\_get function

The table below describes the function spi\_i2s\_flag\_get.

Table 346. spi\_i2s\_flag\_get function

Name	Description
Function name	spi_i2s_flag_get
Function prototype	flag_status spi_i2s_flag_get(spi_type* spi_x, uint32_t spi_i2s_flag);
Function description	Get SPI/I <sup>2</sup> S flags
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4
Input parameter 2	<b>spi_i2s_flag:</b> Select flags Refer to the "spi_i2s_flag" description below for details.
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

### spi\_i2s\_flag

SPI/I<sup>2</sup>S is used to select flags, including:

SPI_I2S_RDBF_FLAG:	SPI/I <sup>2</sup> S receive data buffer full
SPI_I2S_TDBE_FLAG:	SPI/I <sup>2</sup> S transmit data buffer empty
I2S_ACS_FLAG:	I2S Audio channel state (indicating left/right channel)
I2S_TUERR_FLAG:	I2S transmitter underload error
SPI_CCERR_FLAG:	SPI CRC error
SPI_MMERR_FLAG:	SPI master mode error
SPI_I2S_ROERR_FLAG:	SPI/I <sup>2</sup> S receive overflow error
SPI_I2S_BF_FLAG:	SPI/I <sup>2</sup> S busy

### Example:

```
/* get receive data buffer full flag */  
flag_status status;  
status = spi_i2s_flag_get(SPI1, SPI_I2S_RDBF_FLAG);
```

## 5.16.23 spi\_i2s\_interrupt\_flag\_get function

The table below describes the function spi\_i2s\_interrupt\_flag\_get.

**Table 347. spi\_i2s\_interrupt\_flag\_get function**

Name	Description
Function name	spi_i2s_interrupt_flag_get
Function prototype	flag_status spi_i2s_interrupt_flag_get(spi_type* spi_x, uint32_t spi_i2s_flag);
Function description	Get the selected SPI/I <sup>2</sup> S interrupt flag
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4
Input parameter 2	<b>spi_i2s_flag:</b> Select flags Refer to the "spi_i2s_flag" description below for details.
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

### spi\_i2s\_flag

SPI/I<sup>2</sup>S is used to select flags, including:

SPI_I2S_RDBF_FLAG:	SPI/I <sup>2</sup> S receive data buffer full
SPI_I2S_TDBE_FLAG:	SPI/I <sup>2</sup> S transmit data buffer empty
I2S_TUERR_FLAG:	I <sup>2</sup> S transmitter underload error
SPI_CCERR_FLAG:	SPI CRC error
SPI_MMERR_FLAG:	SPI master mode error
SPI_I2S_ROERR_FLAG:	SPI/I <sup>2</sup> S receive overflow error

### Example

```
/* get receive data buffer full flag */
flag_status status;
status = spi_i2s_interrupt_flag_get(SPI1, SPI_I2S_RDBF_FLAG);
```

## 5.16.24 spi\_i2s\_flag\_clear function

The table below describes the function spi\_i2s\_flag\_clear.

Table 348. spi\_i2s\_flag\_clear function

Name	Description
Function name	spi_i2s_flag_clear
Function prototype	void spi_i2s_flag_clear(spi_type* spi_x, uint32_t spi_i2s_flag)
Function description	Clear SPI/I <sup>2</sup> S flags
Input parameter 1	spi_x: select SPI peripheral This parameter can be SPI1, SPI2, SPI3, SPI4
Input parameter 2	<i>spi_i2s_flag</i> : select a flag to clear Refer to the "spi_i2s_flag" description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### spi\_i2s\_flag:

SPI/I<sup>2</sup>S is used for flag selection, including:

- SPI\_I2S\_RDBF\_FLAG: SPI/I<sup>2</sup>S receive data buffer full
- I2S\_TUERR\_FLAG: I2S transmitter underload error
- SPI\_CCERR\_FLAG: SPI CRC error
- SPI\_MMERR\_FLAG: SPI master mode error
- SPI\_I2S\_ROERR\_FLAG: SPI/I<sup>2</sup>S receive overflow error

Note: the SPI\_I2S\_TDBE\_FLAG (SPI/I<sup>2</sup>S transmit data buffer empty), the I2S\_ACS\_FLAG (Audio channel state) and the SPI\_I2S\_BF\_FLAG (SPI/I<sup>2</sup>S busy) are all set and cleared by hardware to indicate communication state, without the intervention of software.

### Example:

```
/* clear receive data buffer full flag */  
spi_i2s_flag_clear (SPI1, SPI_I2S_RDBF_FLAG);
```

## 5.17 SysTick

The SysTick register structure SysTick\_Type is defined in the “core\_cm4.h”:

```
typedef struct
```

```
{
```

```
...
```

```
} SysTick_Type;
```

The table below gives a list of the SysTick registers.

**Table 349. Summary of SysTick registers**

Register	Description
ctrl	Controls status register
load	Reload value register
val	Current counter value register
calib	Calibration register

The table below gives a list of SysTick library functions.

**Table 350. Summary of SysTick library functions**

Function name	Description
systick_clock_source_config	Configure SysTick clock sources
SysTick_Config	Configure SysTick counter reload value and interrupts

### 5.17.1 systick\_clock\_source\_config function

The table below describes the function systick\_clock\_source\_config.

**Table 351. systick\_clock\_source\_config function**

Name	Description
Function name	systick_clock_source_config
Function prototype	void systick_clock_source_config(systick_clock_source_type source);
Function description	Configure SysTick clock source
Input parameter 1	Source: systick clock source
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### source

SYSTICK\_CLOCK\_SOURCE\_AHBCLK\_DIV8: AHB/8 as SysTick clock

SYSTICK\_CLOCK\_SOURCE\_AHBCLK\_NODIV: AHB as SysTick clock

#### Example:

```
/* config systick clock source */
systick_clock_source_config(SYSTICK_CLOCK_SOURCE_AHBCLK_NODIV);
```

## 5.17.2 SysTick\_Config function

The table below describes the function SysTick\_Config.

**Table 352. SysTick\_Config function**

Name	Description
Function name	SysTick_Config
Function prototype	uint32_t SysTick_Config(uint32_t ticks);
Function description	Configure SysTick counter reload value and interrupts
Input parameter 1	Ticks: SysTick counter interrupt reload value
Output parameter	NA
Return value	Return the setting status of this function, success (0) or failure (1)
Required preconditions	NA
Called functions	NA

**Example:**

```
/* config systick reload value and enable interrupt */  
SysTick_Config(1000);
```

## 5.18 SDIO interfaces

SDIO register structure crm\_type is defined in the “at32f403\_sdio.h”:

```
/**  
 * @brief type define sdio register all  
 */  
  
typedef struct  
{  
    ...  
}  
} sdio_type;
```

The table below gives a list of the SDIO registers.

Table 353. SDIO register description

Register	Description
pwrctrl	Power control register
clkctrl	Clock control register
arg	Argument register
cmd	Command register
rspcmd	Command response register
rsp1	Response register 1
rsp2	Response register 2
rsp3	Response register 3
rsp4	Response register 4
dttmr	Data timer register
dtlen	Data length register
dtctrl	Data control register
dtcntr	Data counter register
sts	Status register
intclr	Clear interrupt register
inten	Interrupt mask register
bufcntr	BUF counter register
buf	Data BUF register

The table below gives a list of SDIO library functions.

Table 354. SDIO library functions

Function name	Description
sdio_reset	Reset SDIO peripheral register and control status
sdio_power_set	Set controller power stauts
sdio_power_status_get	Get controller power status
sdio_clock_config	Configure clock parameters
sdio_bus_width_config	Set bus width
sdio_clock_bypass	Enable clock bypass mode
sdio_power_saving_mode_enable	Enable controller power saving mode
sdio_flow_control_enable	Enable flow control mode
sdio_clock_enable	Enable clock
sdio_dma_enable	Enable DMA
sdio_interrupt_enable	Enable interrupts
sdio_flag_get	Get flags
sdio_flag_clear	Clear flags
sdio_command_config	Set command parameters
sdio_command_state_machine_enable	Enable command state machine
sdio_command_response_get	Get command response
sdio_response_get	Return command response
sdio_data_config	Set data parameters
sdio_data_state_machine_enable	Enable data state machine
sdio_data_counter_get	Get data counter
sdio_data_read	Read data (word) from receive FIFO
sdio_buffer_counter_get	Get buffer counter
sdio_data_write	Write data (word) to transmit FIFO
sdio_read_wait_mode_set	Set read wait mode
sdio_read_wait_start	Set read wait start
sdio_read_wait_stop	Set read wait stop
sdio_io_function_enable	Enable IO functional mode
sdio_io_suspend_command_set	Enable IO suspend command

## 5.18.1 sdio\_reset function

The table below describes the function sdio\_reset.

**Table 355. sdio\_reset function**

Name	Description
Function name	sdio_reset
Function prototype	void sdio_reset(sdio_type *sdio_x);
Function description	Reset SDIO peripheral registers and control status
Input parameter 1	sdio_x: select SDIO peripherals, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* reset sdio */
sdio_reset(SDIO1);
```

## 5.18.2 sdio\_power\_set function

The table below describes the function sdio\_power\_set.

**Table 356. sdio\_power\_set function**

Name	Description
Function name	sdio_power_set
Function prototype	void sdio_power_set(sdio_type *sdio_x, sdio_power_state_type power_state);
Function description	Set SDIO power status
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	power_state: indicates power status
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**power\_state**

Set power supply status.

SDIO\_POWER\_ON: Power ON

SDIO\_POWER\_OFF: Power OFF

**Example:**

```
/* sdio power on */
sdio_power_set(SDIO1, SDIO_POWER_ON);
```

### 5.18.3 sdio\_power\_status\_get function

The table below describes the function sdio\_power\_status\_get.

**Table 357. sdio\_power\_status\_get function**

Name	Description
Function name	sdio_power_status_get
Function prototype	sdio_power_state_type sdio_power_status_get(sdio_type *sdio_x);
Function description	Get SDIO power status
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	sdio_power_state_type: indicates power supply status
Required preconditions	NA
Called functions	NA

**Example:**

```
/* check power status */
if(sdio_power_status_get(SDIO1) == SDIO_POWER_OFF)
{
    return SD_REQ_NOT_APPLICABLE;
}
```

### 5.18.4 sdio\_clock\_config function

The table below describes the function sdio\_clock\_config.

**Table 358. sdio\_clock\_config function**

Name	Description
Function name	sdio_clock_config
Function prototype	void sdio_clock_config(sdio_type *sdio_x, uint16_t clk_div, sdio_edge_phase_type clk_edg);
Function description	Configure clock parameters
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	clk_div: set clock division, from 0~0x3FF
Input parameter 3	clk_edg: set clock edge
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**clk\_edg**

Select clock edge.

SDIO\_CLOCK\_EDGE\_RISING: Rising edge

SDIO\_CLOCK\_EDGE\_FALLING: Falling edge

**Example:**

```
/* config sdio clock divide and edge phase */
sdio_clock_config(SDIO1, 0x2, SDIO_CLOCK_EDGE_FALLING);
```

## 5.18.5 sdio\_bus\_width\_config function

The table below describes the function sdio\_bus\_width\_config.

**Table 359. sdio\_bus\_width\_config function**

Name	Description
Function name	sdio_bus_width_config
Function prototype	void sdio_bus_width_config(sdio_type *sdio_x, sdio_bus_width_type width);
Function description	Configure SDIO bus width
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	Width: select SDIO bus width
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### width

Select bus width.

SDIO\_BUS\_WIDTH\_D1: 1-bit

SDIO\_BUS\_WIDTH\_D4: 4-bit

SDIO\_BUS\_WIDTH\_D8: 8-bit

### Example:

```
/* config sdio bus width */
sdio_bus_width_config(SDIOx, SDIO_BUS_WIDTH_D1);
```

## 5.18.6 sdio\_clock\_bypass function

The table below describes the function sdio\_clock\_bypass.

**Table 360. sdio\_clock\_bypass function**

Name	Description
Function name	sdio_clock_bypass
Function prototype	void sdio_clock_bypass(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable SDIO clock bypass mode
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	new_state: Bypass ON (TRUE), Bypass OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### Example:

```
/* disable clock bypass */
sdio_clock_bypass(SDIO1, FALSE);
```

## 5.18.7 sdio\_power\_saving\_mode\_enable function

The table below describes the function sdio\_power\_saving\_mode\_enable.

**Table 361. sdio\_power\_saving\_mode\_enable function**

Name	Description
Function name	sdio_power_saving_mode_enable
Function prototype	void sdio_power_saving_mode_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable SDIO power saving mode
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	new_state: ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* disable power saving mode */
sdio_power_saving_mode_enable(SDIO1, FALSE);
```

## 5.18.8 sdio\_flow\_control\_enable function

The table below describes the function sdio\_flow\_control\_enable.

**Table 362. sdio\_flow\_control\_enable function**

Name	Description
Function name	sdio_flow_control_enable
Function prototype	void sdio_flow_control_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable SDIO flow control mode
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	new_state: ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* disable flow control */
sdio_flow_control_enable(SDIO1, FALSE);
```

## 5.18.9 sdio\_clock\_enable function

The table below describes the function sdio\_clock\_enable.

**Table 363. sdio\_clock\_enable function**

Name	Description
Function name	sdio_clock_enable
Function prototype	void sdio_clock_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable SDIO clock
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	new_state: ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable to output sdio_ck */  
sdio_clock_enable(SDIO1, TRUE);
```

## 5.18.10 sdio\_dma\_enable function

The table below describes the function sdio\_dma\_enable.

**Table 364. sdio\_dma\_enable function**

Name	Description
Function name	sdio_dma_enable
Function prototype	void sdio_dma_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable SDIO DMA mode
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	new_state: ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable sdio dma */  
sdio_dma_enable(SDIO1, TRUE);
```

## 5.18.11 sdio\_interrupt\_enable function

The table below describes the function `sdio_interrupt_enable`.

**Table 365. `sdio_interrupt_enable` function**

Name	Description
Function name	<code>sdio_interrupt_enable</code>
Function prototype	<code>void sdio_interrupt_enable(sdio_type *sdio_x, uint32_t int_opt, confirm_state new_state);</code>
Function description	Enable SDIO interrupts
Input parameter 1	<code>sdio_x</code> : select SDIO peripheral, such as SDIO1
Input parameter 2	<code>int_opt</code> : select interrupt type
Input parameter 3	<code>new_state</code> : ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### `int_opt`

Select interrupts.

<code>SDIO_CMDFAIL_INT:</code>	Command CRC fail interrupt
<code>SDIO_DTFAIL_INT:</code>	Data CRC fail interrupt
<code>SDIO_CMDTIMEOUT_INT:</code>	Command timeout interrupt
<code>SDIO_DTTIMEOUT_INT:</code>	Data timeout interrupt
<code>SDIO_TXERRU_INT:</code>	TxBUF underrun error interrupt
<code>SDIO_RXERRO_INT:</code>	RxBUF overrun error interrupt
<code>SDIO_CMDRSPCMPL_INT:</code>	Command response received interrupt
<code>SDIO_CMDCMPL_INT:</code>	Command transfer complete interrupt
<code>SDIO_DTCMP_INT:</code>	Data transfer complete interrupt
<code>SDIO_SBITERR_INT:</code>	Start bit error interrupt
<code>SDIO_DTBLOCKCMPL_INT:</code>	Data block transfer complete interrupt
<code>SDIO_DOCMD_INT:</code>	Command transfer in-progress interrupt
<code>SDIO_DOTX_INT:</code>	Data transfer in-progress interrupt
<code>SDIO_DORX_INT:</code>	Data receive in-progress interrupt
<code>SDIO_TXBUFH_INT:</code>	Transmit buffer half-empty
<code>SDIO_RXBUFH_INT:</code>	Receive buffer half-empty
<code>SDIO_TXBUFF_INT:</code>	Transmit buffer full
<code>SDIO_RXBUFF_INT:</code>	Receive buffer full
<code>SDIO_TXBUFE_INT:</code>	Transmit buffer empty
<code>SDIO_RXBUFE_INT:</code>	Receive buffer empty
<code>SDIO_TXBUF_INT:</code>	Data available in transmit BUF
<code>SDIO_RXBUF_INT:</code>	Data available in receive BUF
<code>SDIO_SDIOIF_INT:</code>	SD I/O receive interrupt

### Example:

```
/* disable interrupt */
sdio_interrupt_enable(SDIO1, (SDIO_DTFAIL_INT | SDIO_DTTIMEOUT_INT | \
                           SDIO_DTCMP_INT | SDIO_TXBUFFH_INT | SDIO_RXBUFH_INT | \
                           SDIO_TXERRU_INT | SDIO_RXERRO_INT | SDIO_SBITERR_INT), FALSE);
```

## 5.18.12 sdio\_flag\_get function

The table below describes the function `sdio_flag_get`.

**Table 366. `sdio_flag_get` function**

Name	Description
Function name	<code>sdio_flag_get</code>
Function prototype	<code>flag_status sdio_flag_get(sdio_type *sdio_x, uint32_t flag);</code>
Function description	Get flag status
Input parameter 1	<code>sdio_x</code> : select SDIO peripheral, such as SDIO1
Input parameter 2	Flag: interrupt type
Output parameter	NA
Return value	<code>flag_status</code> : SET or RESET
Required preconditions	NA
Called functions	NA

### flag

Select flags.

<code>SDIO_CMDFAIL_FLAG</code> :	Command CRC failure
<code>SDIO_DTFAIL_FLAG</code> :	Data CRC failure
<code>SDIO_CMDTIMEOUT_FLAG</code> :	Command timeout
<code>SDIO_DTTIMEOUT_FLAG</code> :	Data timeout
<code>SDIO_TXERRU_FLAG</code> :	TxBUF underrun error
<code>SDIO_RXERRO_FLAG</code> :	RxBUF overrun error
<code>SDIO_CMDRSPCMPL_FLAG</code> :	Command response received
<code>SDIO_CMDCMPL_FLAG</code> :	Command transfer complete
<code>SDIO_DTCMP_FLAG</code> :	Data transfer complete
<code>SDIO_SBITERR_FLAGv</code>	Start bit error
<code>SDIO_DTBLKCMPL_FLAG</code> :	Data block transfer complete
<code>SDIO_DOCMD_FLAG</code> :	Command transfer in-progress
<code>SDIO_DOTX_FLAG</code> :	Data transfer in-progress
<code>SDIO_DORX_FLAG</code> :	Data receive in-progress
<code>SDIO_TXBUFH_FLAG</code> :	Transmit buffer half-empty
<code>SDIO_RXBUFH_FLAG</code> :	Receive buffer half-empty
<code>SDIO_TXBUFF_FLAG</code> :	Transmit buffer full
<code>SDIO_RXBUFF_FLAG</code> :	Receive buffer full
<code>SDIO_TXBUFE_FLAG</code> :	Transmit buffer empty
<code>SDIO_RXBUFE_FLAG</code> :	Receive buffer empty
<code>SDIO_TXBPF_FLAG</code> :	Data available in transmit BUF
<code>SDIO_RXBPF_FLAG</code> :	Data available in receive BUF
<code>SDIO_SDIOIF_FLAG</code> :	SD I/O receive

### Example:

```
/* check dttimeout flag */
if(sdio_flag_get(SDIOx, SDIO_DTTIMEOUT_FLAG) != RESET)
{
}
```

### 5.18.13 sdio\_interrupt\_flag\_get function

The table below describes the function `sdio_interrupt_flag_get`.

**Table 367. `sdio_interrupt_flag_get` function**

Name	Description
Function name	<code>sdio_interrupt_flag_get</code>
Function prototype	<code>flag_status sdio_interrupt_flag_get(sdio_type *sdio_x, uint32_t flag);</code>
Function description	Check whether the selected interrupt flag is set or not
Input parameter 1	<code>sdio_x</code> : select SDIO peripheral, such as SDIO1
Input parameter 2	<code>flag</code> : interrupt type
Output parameter	NA
Return value	<code>flag_status</code> : SET or RESET
Required preconditions	NA
Called functions	NA

#### flag

Select a flag

<code>SDIO_CMDFAIL_FLAG:</code>	Command CRC failure
<code>SDIO_DTFAIL_FLAG:</code>	Data CRC failure
<code>SDIO_CMDTIMEOUT_FLAG:</code>	Command timeout
<code>SDIO_DTTIMEOUT_FLAG:</code>	Data timeout
<code>SDIO_TXERRU_FLAG:</code>	TxBUF underrun error
<code>SDIO_RXERRO_FLAG:</code>	RxBUF overrun error
<code>SDIO_CMDRSPCMPL_FLAG:</code>	Command response received
<code>SDIO_CMDCMPL_FLAG:</code>	Command transfer complete
<code>SDIO_DTCMP_FLAG:</code>	Data transfer complete
<code>SDIO_SBITERR_FLAG:</code>	Start bit error
<code>SDIO_DTBLKCMPL_FLAG:</code>	Data block transfer complete
<code>SDIO_DOCMD_FLAG:</code>	Command transfer in-progress
<code>SDIO_DOTX_FLAG:</code>	Data transfer in-progress
<code>SDIO_DORX_FLAG:</code>	Data receive in-progress
<code>SDIO_TXBUFH_FLAG:</code>	Transmit BUF half-empty
<code>SDIO_RXBUFH_FLAG:</code>	Receive BUF half-empty
<code>SDIO_TXBUFF_FLAG:</code>	Transmit BUF full
<code>SDIO_RXBUFF_FLAG:</code>	Receive BUF full
<code>SDIO_TXBUFE_FLAG:</code>	Transmit BUF empty
<code>SDIO_RXBUFE_FLAG:</code>	Receive BUF empty
<code>SDIO_TXBUFF_FLAG:</code>	Data available in transmit BUF
<code>SDIO_RXBUFF_FLAG:</code>	Data available in receive BUF
<code>SDIO_SDIOIF_FLAG:</code>	SD I/O receive

#### Example

```
/* check dttimeout interrupt flag */
if(sdio_interrupt_flag_get(SDIOx, SDIO_DTTIMEOUT_FLAG) != RESET)
{
}
```

## 5.18.14 sdio\_flag\_clear function

The table below describes the function sdio\_flag\_clear.

Table 368. sdio\_flag\_clear function

Name	Description
Function name	sdio_flag_clear
Function prototype	void sdio_flag_clear(sdio_type *sdio_x, uint32_t flag);
Function description	Clear flags
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	Flag: interrupt type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### flag

Select flags.

SDIO_CMDFAIL_FLAG:	Command CRC failure
SDIO_DTFAIL_FLAG:	Data CRC failure
SDIO_CMDTIMEOUT_FLAG:	Command timeout
SDIO_DTTIMEOUT_FLAG:	Data timeout
SDIO_TXERRU_FLAG:	TxBUF underrun error
SDIO_RXERRO_FLAG:	RxBUF overrun error
SDIO_CMDRSPCMPL_FLAG:	Command response received
SDIO_CMDCMPL_FLAG:	Command transfer complete
SDIO_DTCMP_FLAG:	Data transfer complete
SDIO_SBITERR_FLAGv	Start bit error
SDIO_DTBLKCMPL_FLAG:	Data block transfer complete
SDIO_SDIOIF_FLAG:	SD I/O receive

### Example:

```
/* clear flags */  
#define SDIO_STATIC_FLAGS ((uint32_t)0x000005FF)  
sdio_flag_clear(SDIO1, SDIO_STATIC_FLAGS);
```

## 5.18.15 sdio\_command\_config function

The table below describes the function sdio\_command\_config.

**Table 369. sdio\_command\_config function**

Name	Description
Function name	sdio_command_config
Function prototype	void sdio_command_config(sdio_type *sdio_x, sdio_command_struct_type *command_struct);
Function description	Configure command parameters
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	command_struct: sdio_command_struct_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### command\_struct

sdio\_command\_struct\_type is defined in the at32f403\_sdio.h:

```
typedef struct
{
    uint32_t           argument;
    uint8_t            cmd_index;
    sdio_reponse_type rsp_type;
    sdio_wait_type    wait_type;
} sdio_command_struct_type;
```

### argument

This command parameter is part of command information sent to the card, depending on the type of commands.

### cmd\_index

Indicates command index sent.

### rsp\_type

This is a response parameter, depending on the type of commands, including:

SDIO_RESPONSE_NO:	No response
SDIO_RESPONSE_SHORT:	Short response
SDIO_RESPONSE_LONG:	Long response

### wait\_type

This a wait parameter, depending on the types of commands, including:

SDIO_WAIT_FOR_NO:	No response
SDIO_WAIT_FOR_INT:	Wait for interrupt requests
SDIO_WAIT_FOR_PEND:	Wait for the end of transfer

### Example:

```
/* send cmd16, set block length */
sdio_command_struct_type sdio_command_init_struct;
sdio_command_init_struct.argument = (uint32_t)8;
sdio_command_init_struct.cmd_index = SD_CMD_SET_BLOCKLEN;
sdio_command_init_struct.rsp_type = SDIO_RESPONSE_SHORT;
sdio_command_init_struct.wait_type = SDIO_WAIT_FOR_NO;
```

```
/* sdio command config */
sdio_command_config(SDIOx, &sdio_command_init_struct);
```

### 5.18.16 sdio\_command\_state\_machine\_enable function

The table below describes the function sdio\_command\_state\_machine\_enable.

**Table 370. sdio\_command\_state\_machine\_enable function**

Name	Description
Function name	sdio_command_state_machine_enable
Function prototype	void sdio_command_state_machine_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable command state machine
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	new_state: ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable ccsm */
sdio_command_state_machine_enable(SDIO1, TRUE);
```

### 5.18.17 sdio\_command\_response\_get function

The table below describes the function sdio\_command\_response\_get.

**Table 371. sdio\_command\_response\_get function**

Name	Description
Function name	sdio_command_response_get
Function prototype	uint8_t sdio_command_response_get(sdio_type *sdio_x);
Function description	Get command index corresponding to the received command response
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	uint8_t: return command index corresponding to command response
Required preconditions	NA
Called functions	NA

**Example:**

```
/* get response of command index */
uint8_t rsp_cmd = 0;
rsp_cmd = sdio_command_response_get(SDIO1);
```

## 5.18.18 sdio\_response\_get function

The table below describes the function sdio\_response\_get.

**Table 372. sdio\_response\_get function**

Name	Description
Function name	sdio_response_get
Function prototype	uint32_t sdio_response_get(sdio_type *sdio_x, sdio_rsp_index_type reg_index);
Function description	Get command response
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	reg_index: indicate response register index, including 1/2/3/4 response registers
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### reg\_div

Response register index.

- SDIO\_RSP1\_INDEX: Response register 1
- SDIO\_RSP2\_INDEX: Response register 2
- SDIO\_RSP3\_INDEX: Response register 3
- SDIO\_RSP4\_INDEX: Response register 4

### Example:

```
/* get response register1 */
response = sdio_response_get(SDIO1, SDIO_RSP1_INDEX);
```

## 5.18.19 sdio\_data\_config function

The table below describes the function sdio\_data\_config.

**Table 373. sdio\_data\_config function**

Name	Description
Function name	sdio_data_config
Function prototype	void sdio_data_config(sdio_type *sdio_x, sdio_data_struct_type *data_struct);
Function description	Configure SDIO data parameters
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	data_struct: sdio_data_struct_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### data\_struct

The sdio\_data\_struct\_type is defined in the at32f403\_sdio.h:

typedef struct

```
{
    uint32_t                  timeout;
    uint32_t                  data_length;
    sdio_block_size_type      block_size;
```

```
    sdio_transfer_mode_type      transfer_mode;
    sdio_transfer_direction_type transfer_direction;
} sdio_data_struct_type;
```

**timeout**

Indicates timeout period value for data transfer, counting based on bus clock

**data\_length**

Indicates the length of data to transfer

**block\_size**

Indicates block size, including:

SDIO_DATA_BLOCK_SIZE_1B:	1-bit
SDIO_DATA_BLOCK_SIZE_2B:	2-bit
SDIO_DATA_BLOCK_SIZE_4B:	4-bit
SDIO_DATA_BLOCK_SIZE_8B:	8-bit
SDIO_DATA_BLOCK_SIZE_16B:	16-bit
SDIO_DATA_BLOCK_SIZE_32B:	32-bit
SDIO_DATA_BLOCK_SIZE_64B:	64-bit
SDIO_DATA_BLOCK_SIZE_128B:	128-bit
SDIO_DATA_BLOCK_SIZE_256B:	256-bit
SDIO_DATA_BLOCK_SIZE_512B:	512-bit
SDIO_DATA_BLOCK_SIZE_1024B:	1024-bit
SDIO_DATA_BLOCK_SIZE_2048B:	2048-bit
SDIO_DATA_BLOCK_SIZE_4096B:	4096-bit
SDIO_DATA_BLOCK_SIZE_8192B:	8192-bit
SDIO_DATA_BLOCK_SIZE_16384B:	16384-bit

**transfer\_mode**

Select data transfer mode

SDIO\_DATA\_BLOCK\_TRANSFER: Data block transfer

SDIO\_DATA\_STREAM\_TRANSFER: Data stream transfer

**transfer\_direction**

Select data transfer direction.

SDIO\_DATA\_TRANSFER\_TO\_CARD: Controller to card

SDIO\_DATA\_TRANSFER\_TO\_CONTROLLER: Card to controller

**Example:**

```
sdio_data_struct_type sdio_data_init_struct;
sdio_data_init_struct.block_size = SDIO_DATA_BLOCK_SIZE_512B;
sdio_data_init_struct.data_length = 8 ;
sdio_data_init_struct.timeout = SD_DATATIMEOUT ;
sdio_data_init_struct.transfer_direction = SDIO_DATA_TRANSFER_TO_CARD;
sdio_data_init_struct.transfer_mode = SDIO_DATA_BLOCK_TRANSFER;
/* config sdio data */
sdio_data_config(SDIO1, &sdio_data_init_struct);
```

## 5.18.20 sdio\_data\_state\_machine\_enable function

The table below describes the function `sdio_data_state_machine_enable`.

**Table 374. `sdio_data_state_machine_enable` function**

Name	Description
Function name	<code>sdio_data_state_machine_enable</code>
Function prototype	<code>void sdio_data_state_machine_enable(sdio_type *sdio_x, confirm_state new_state);</code>
Function description	Enable data state machine
Input parameter 1	<code>sdio_x</code> : select SDIO peripheral, such as SDIO1
Input parameter 2	<code>new_state</code> : ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable dcsm */
sdio_data_state_machine_enable(SDIO1, TRUE);
```

## 5.18.21 sdio\_data\_counter\_get function

The table below describes the function `sdio_data_counter_get`.

**Table 375. `sdio_data_counter_get` function**

Name	Description
Function name	<code>sdio_data_counter_get</code>
Function prototype	<code>uint32_t sdio_data_counter_get(sdio_type *sdio_x);</code>
Function description	Get SDIO data counter
Input parameter 1	<code>sdio_x</code> : select SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	<code>uint32_t</code> : return the data count to be sent
Required preconditions	NA
Called functions	NA

**Example:**

```
/* get data counter */
uint32_t count = 0;
count = sdio_data_counter_get (SDIO1);
```

## 5.18.22 sdio\_data\_read function

The table below describes the function sdio\_data\_read.

**Table 376. sdio\_data\_read function**

Name	Description
Function name	sdio_data_read
Function prototype	uint32_t sdio_data_read(sdio_type *sdio_x);
Function description	Read a word data from receive FIFO
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	NA
Input parameter 3	NA
Output parameter	NA
Return value	uint32_t: read data (word)
Required preconditions	NA
Called functions	NA

**Example:**

```
/* read data */
uint32_t data = 0;
data = sdio_data_read(SDIO1);
```

## 5.18.23 sdio\_buffer\_counter\_get function

The table below describes the function sdio\_buffer\_counter\_get.

**Table 377. sdio\_buffer\_counter\_get function**

Name	Description
Function name	sdio_buffer_counter_get
Function prototype	uint32_t sdio_buffer_counter_get(sdio_type *sdio_x);
Function description	Get the number of data to read/write from/to BUFF
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* get buffer count */
uint32_t count = 0;
count = sdio_buffer_counter_get(SDIO1);
```

## 5.18.24 sdio\_data\_write function

The table below describes the function sdio\_data\_write.

**Table 378. sdio\_data\_write function**

Name	Description
Function name	sdio_data_write
Function prototype	void sdio_data_write(sdio_type *sdio_x, uint32_t data);
Function description	Write a word data to transmit FIFO
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* write data */
uint32_t data = 0x11223344;
sdio_data_write(SDIO1, data);
```

## 5.18.25 sdio\_read\_wait\_mode\_set function

The table below describes the function sdio\_read\_wait\_mode\_set.

**Table 379. sdio\_read\_wait\_mode\_set function**

Name	Description
Function name	sdio_read_wait_mode_set
Function prototype	void sdio_read_wait_mode_set(sdio_type *sdio_x, sdio_read_wait_mode_type mode);
Function description	Set read wait mode
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	Mode: read wait mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**mode**

SDIO\_READ\_WAIT\_CONTROLLED\_BY\_D2: Read wait controlled by DATA Line2

SDIO\_READ\_WAIT\_CONTROLLED\_BY\_CK: Read wait controlled by clock line

**Example:**

```
/* config read wait mode */
sdio_read_wait_mode_set(SDIO1, SDIO_READ_WAIT_CONTROLLED_BY_D2);
```

## 5.18.26 sdio\_read\_wait\_start function

The table below describes the function sdio\_read\_wait\_start.

**Table 380. sdio\_read\_wait\_start function**

Name	Description
Function name	sdio_read_wait_start
Function prototype	void sdio_read_wait_start(sdio_type *sdio_x, confirm_state new_state);
Function description	Read wait start
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	new_state: ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Calling this function ON indicates that read wait mode is enabled, OFF no effect.

**Example:**

```
/* start read wait mode */  
sdio_read_wait_start (SDIO1, TRUE);
```

## 5.18.27 sdio\_read\_wait\_stop function

The table below describes the function sdio\_read\_wait\_stop.

**Table 381. sdio\_read\_wait\_stop function**

Name	Description
Function name	sdio_read_wait_stop
Function prototype	void sdio_read_wait_stop(sdio_type *sdio_x, confirm_state new_state);
Function description	Read wait stop
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	new_state: ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Calling this function “ON” indicates that read wait mode is disabled, “OFF” read wait mode continues.

**Example:**

```
/* stop read wait mode */  
sdio_read_wait_stop (SDIO1, TRUE);
```

## 5.18.28 sdio\_io\_function\_enable function

The table below describes the function sdio\_io\_function\_enable.

**Table 382. sdio\_io\_function\_enable function**

Name	Description
Function name	sdio_io_function_enable
Function prototype	void sdio_io_function_enable(sdio_type *sdio_x, confirm_state new_state);
Function description	Enable SDIO IO functional mode
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	new_state: ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable sdio IO mode */  
sdio_io_function_enable (SDIO1, TRUE);
```

## 5.18.29 sdio\_io\_suspend\_command\_set function

The table below describes the function sdio\_io\_suspend\_command\_set.

**Table 383. sdio\_io\_suspend\_command\_set function**

Name	Description
Function name	sdio_io_suspend_command_set
Function prototype	void sdio_io_suspend_command_set(sdio_type *sdio_x, confirm_state new_state);
Function description	Set SDIO IO suspend command
Input parameter 1	sdio_x: select SDIO peripheral, such as SDIO1
Input parameter 2	new_state: ON (TRUE), OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* send suspend command */  
sdio_io_suspend_command_set (SDIO1, TRUE);
```

## 5.19 TMR

The TMR register structure tmr\_type is defined in the "at32f403\_tmr.h":

```
/*
 * @brief type define tmr register all
 */
typedef struct
{
} tmr_type;
```

The table below gives a list of the TMR registers.

**Table 384. Summary of TMR registers**

Register	Description
ctrl1	TMR control register 1
ctrl2	TMR control register 2
stctrl	TMR slave timer control register
iden	TMR DMA/ interrupt enable register
ists	TMR interrupt status register
swevt	TMR software event register
cm1	TMR channel mode register 1
cm2	TMR channel mode register 2
cctrl	TMR channel control register
cval	TMR counter value register
div	TMR division register
pr	TMR period register
rpr	TMR repetition period channel
c1dt	TMR channel 1 data register
c2dt	TMR channel 2 data register
c3dt	TMR channel 3 data register
c4dt	TMR channel 4 data register
brk	TMR break register
dmactrl	TMR DMA control register
dmadt	TMR DMA data register

The table below gives a list of TMR library functions.

Table 385. Summary of TMR library functions

Function name	Description
tmr_reset	TMR is reset by CRM reset register
tmr_counter_enable	Enable or disable TMR
tmr_output_default_para_init	Initialize TMR output default parameters
tmr_input_default_para_init	Initialize TMR input default parameters
tmr_brkdt_default_para_init	Initialize TMR brkdt default parameters
tmr_base_init	Initialize TMR period and division
tmr_clock_source_div_set	Set TMR clock source frequency division factor
tmr_cnt_dir_set	Set TMR counter direction
tmr_repetition_counter_set	Set repetition period register
tmr_counter_value_set	Set TMR counter value
tmr_counter_value_get	Get TMR counter value
tmr_div_value_set	Set TMR division value
tmr_div_value_get	Get TMR division value
tmr_output_channel_config	Configure TMR output channels
tmr_output_channel_mode_select	Select TMR output channel mode
tmr_period_value_set	Set TMR period value
tmr_period_value_get	Get TMR period value
tmr_channel_value_set	Set TMR channel value
tmr_channel_value_get	Get TMR channel value
tmr_period_buffer_enable	Enable or disable TMR periodic buffer
tmr_output_channel_buffer_enable	Enable or disable TMR output channel buffer
tmr_output_channel_immediately_set	TMR output channel enable immediately
tmr_output_channel_switch_set	Set TMR output channel switch
tmr_one_cycle_mode_enable	Enable or disable TMR one-cycle mode
tmr_32_bit_function_enable	Enable or disable TMR 32-bit function (plus mode)
tmr_overflow_request_source_set	Select TMR overflow event source
tmr_overflow_event_disable	Enable or disable TMR overflow event generation
tmr_channel_enable	Enable or disable TMR channel
tmr_input_channel_filter_set	Set TMR input channel filter
tmr_pwm_input_config	Configure TMR pwm input
tmr_channel1_input_select	Select TMR channel 1 input
tmr_input_channel_divider_set	Set TMR input channel divider
tmr_primary_mode_select	Select TMR master mode
tmr_sub_mode_select	Select TMR slave timer mode
tmr_channel_dma_select	Select TMR channel DMA request source
tmr_hall_select	Select TMR hall mode
tmr_channel_buffer_enable	Enable or disable TMR channel buffer
tmr_trigger_input_select	Select TMR slave timer trigger input
tmr_sub_sync_mode_set	Set TMR slave timer synchronization mode
tmr_dma_request_enable	Enable or disable TMR DMA request
tmr_interrupt_enable	Enable or disable TMR interrupt
tmr_flag_get	Get TMR flags
tmr_flag_clear	Clear TMR flags

tmr_event_sw_trigger	Software trigger TMR event
tmr_output_enable	Enable or disable TMR output
tmr_internal_clock_set	Set TMR internal clock
tmr_output_channel_polarity_set	Set TMR output channel polarity
tmr_external_clock_config	Set TMR external clock
tmr_external_clock_mode1_config	Set TMR external clock mode 1
tmr_external_clock_mode2_config	Set TMR external clock mode 2
tmr_encoder_mode_config	Set TMR encode mode
tmr_force_output_set	Set TMR forced output
tmr_dma_control_config	Set TMR DMA control
tmr_brkdt_config	Set TMR break mode and dead-time

### 5.19.1 tmr\_reset function

The table below describes the function tmr\_reset.

Table 386. tmr\_reset function

Name	Description
Function name	tmr_reset
Function prototype	void tmr_reset(tmr_type *tmr_x);
Function description	TMR is reset by CRM reset register.
Input parameter	tmr_x: select TMR peripheral, including: TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset();

**Example:**

```
tmr_reset(TMR1);
```

### 5.19.2 tmr\_counter\_enable function

The table below describes the function tmr\_counter\_enable.

Table 387. tmr\_counter\_enable function

Name	Description
Function name	tmr_counter_enable
Function prototype	void tmr_counter_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR
Input parameter 1	tmr_x: select TMR peripheral, including: TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	new_state: indicates counter status, ON (TRUE) or OFF (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

tmr_counter_enable(TMR1, TRUE);
---------------------------------

### 5.19.3 tmr\_output\_default\_para\_init function

The table below describes the function tmr\_output\_default\_para\_init.

**Table 388. tmr\_output\_default\_para\_init function**

Name	Description
Function name	tmr_output_default_para_init
Function prototype	void tmr_output_default_para_init(tmr_output_config_type *tmr_output_struct);
Function description	Initialize tmr output default parameters
Input parameter	tmr_output_struct: tmr_output_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the function tmr\_output\_struct.

**Table 389. tmr\_output\_struct default values**

Member	Default values
oc_mode	TMR_OUTPUT_CONTROL_OFF
oc_idle_state	FALSE
occ_idle_state	FALSE
oc_polarity	TMR_OUTPUT_ACTIVE_HIGH
occ_polarity	TMR_OUTPUT_ACTIVE_HIGH
oc_output_state	FALSE
occ_output_state	FALSE

**Example:**

tmr_output_config_type tmr_output_struct;
tmr_output_default_para_init(&tmr_output_struct);

### 5.19.4 tmr\_input\_default\_para\_init function

The table below describes the function tmr\_input\_default\_para\_init.

**Table 390. tmr\_input\_default\_para\_init function**

Name	Description
Function name	tmr_input_default_para_init
Function prototype	void tmr_input_default_para_init(tmr_input_config_type *tmr_input_struct);
Function description	Initialize TMR input default parameters
Input parameter	tmr_input_struct: tmr_input_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the function tmr\_input\_struct.

**Table 391. tmr\_input\_struct default values**

Member	Default values
input_channel_select	TMR_SELECT_CHANNEL_1
input_polarity_select	TMR_INPUT_RISING_EDGE
input_mapped_select	TMR_CC_CHANNEL_MAPPED_DIRECT
input_filter_value	0x0

**Example:**

```
tmr_input_config_type tmr_input_struct;
tmr_input_default_para_init(&tmr_input_struct);
```

### 5.19.5 tmr\_brkdt\_default\_para\_init function

The table below describes the function tmr\_brkdt\_default\_para\_init.

**Table 392. tmr\_brkdt\_default\_para\_init function**

Name	Description
Function name	tmr_brkdt_default_para_init
Function prototype	void tmr_brkdt_default_para_init(tmr_brkdt_config_type *tmr_brkdt_struct);
Function description	Initialize TMR brkdt default parameters
Input parameter	tmr_brkdt_struct: tmr_brkdt_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below describes the default values of members of the function tmr\_brkdt\_struct.

**Table 393. tmr\_brkdt\_struct default values**

Member	Default values
deadtime	0x0
brk_polarity	TMR_BRK_INPUT_ACTIVE_LOW
wp_level	TMR_WP_OFF
auto_output_enable	FALSE
fcsoen_state	FALSE
fcsodis_state	FALSE
brk_enable	FALSE

**Example:**

```
tmr_brkdt_config_type tmr_brkdt_struct;
tmr_brkdt_default_para_init(&tmr_brkdt_struct);
```

## 5.19.6 tmr\_base\_init function

The table below describes the function tmr\_base\_init.

**Table 394. tmr\_base\_init function**

Name	Description
Function name	tmr_base_init
Function prototype	void tmr_base_init(tmr_type* tmr_x, uint32_t tmr_pr, uint32_t tmr_div);
Function description	Initialize TMR period and division
Input parameter 1	tmr_x: TMR peripheral including: TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_pr: timer period value, for 16-bit timer, it can be 0x0000~0xFFFF; for 32-bit timer, it can be 0x0000_0000~0xFFFF_FFFF
Input parameter 3	tmr_div: timer division value, 0x0000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

tmr_base_init(TMR1, 0xFFFF, 0xFFFF);
--------------------------------------

## 5.19.7 tmr\_clock\_source\_div\_set function

The table below describes the function tmr\_clock\_source\_div\_set.

**Table 395. tmr\_clock\_source\_div\_set function**

Name	Description
Function name	tmr_clock_source_div_set
Function prototype	void tmr_clock_source_div_set(tmr_type *tmr_x, tmr_clock_division_type tmr_clock_div);
Function description	Set TMR clock source division factor
Input parameter 1	tmr_x: TMR peripheral, including TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_clock_div: timer clock source frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**tmr\_clock\_div**

Select TMR clock source frequency division factor

TMR\_CLOCK\_DIV1: Divided by 1

TMR\_CLOCK\_DIV2: Divided by 2

TMR\_CLOCK\_DIV4: Divided by 4

**Example:**

tmr_clock_source_div_set(TMR1, TMR_CLOCK_DIV4);
---

## 5.19.8 tmr\_cnt\_dir\_set function

The table below describes the function tmr\_cnt\_dir\_set.

**Table 396. tmr\_cnt\_dir\_set function**

Name	Description
Function name	tmr_cnt_dir_set
Function prototype	void tmr_cnt_dir_set(tmr_type *tmr_x, tmr_count_mode_type tmr_cnt_dir);
Function description	Set TMR counter direction
Input parameter 1	tmr_x: indicates the selected TMR peripheral, including TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_cnt_dir: timer counting direction
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

tmr\_cnt\_dir

Select timer counting direction.

TMR\_COUNT\_UP: Up counting

TMR\_COUNT\_DOWN: Down counting

TMR\_COUNT\_TWO\_WAY\_1: Center-aligned mode (up/down counting) 1

TMR\_COUNT\_TWO\_WAY\_2: Center-aligned mode (up/down counting) 2

TMR\_COUNT\_TWO\_WAY\_3: Center-aligned mode (up/down counting) 3

**Example:**

```
tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);
```

## 5.19.9 tmr\_repetition\_counter\_set function

The table below describes the function tmr\_repetition\_counter\_set.

**Table 397. tmr\_repetition\_counter\_set function**

Name	Description
Function name	tmr_repetition_counter_set
Function prototype	void tmr_repetition_counter_set(tmr_type *tmr_x, uint8_t tmr_rpr_value);
Function description	Set repetition period register (rpr)
Input parameter 1	tmr_x: TMR peripheral This parameter can be TMR1, TMR8, TMR15
Input parameter 2	tmr_rpr_value: timer repetition period value, it can be 0x00~0xFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
tmr_repetition_counter_set(TMR1, 0x10);
```

### 5.19.10 tmr\_counter\_value\_set function

The table below describes the function tmr\_counter\_value\_set.

**Table 398. tmr\_counter\_value\_set function**

Name	Description
Function name	tmr_counter_value_set
Function prototype	void tmr_counter_value_set(tmr_type *tmr_x, uint32_t tmr_cnt_value);
Function description	Set TMR counter value
Input parameter 1	tmr_x: indicates the selected TMR peripheral It can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_cnt_value: timer counter value, for 16-bit timer, it can be 0x0000~0xFFFF; For 32-bit timer, it can be 0x0000_0000~0xFFFF_FFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
tmr_counter_value_set(TMR1, 0xFFFF);
```

### 5.19.11 tmr\_counter\_value\_get function

The table below describes the function tmr\_counter\_value\_get.

**Table 399. tmr\_counter\_value\_get function**

Name	Description
Function name	tmr_counter_value_get
Function prototype	uint32_t tmr_counter_value_get(tmr_type *tmr_x);
Function description	Get TMR counter value
Input parameter	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Output parameter	NA
Return value	Timer counter value
Required preconditions	NA
Called functions	NA

**Example:**

```
uint32_t counter_value;
counter_value = tmr_counter_value_get(TMR1);
```

## 5.19.12 tmr\_div\_value\_set function

The table below describes the function tmr\_div\_value\_set.

**Table 400. tmr\_div\_value\_set function**

Name	Description
Function name	tmr_div_value_set
Function prototype	void tmr_div_value_set(tmr_type *tmr_x, uint32_t tmr_div_value);
Function description	Set TMR frequency division value
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_div_value: timer frequency division value. For 16-bit timer, it can be0x0000~0xFFFF; for 32-bit timer, it can be 0x0000_0000~0xFFFF_FFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
tmr_div_value_set(TMR1, 0xFFFF);
```

## 5.19.13 tmr\_div\_value\_get function

The table below describes the function tmr\_div\_value\_get.

**Table 401. tmr\_div\_value\_get function**

Name	Description
Function name	tmr_div_value_get
Function prototype	uint32_t tmr_div_value_get(tmr_type *tmr_x);
Function description	Get TMR frequency division value
Input parameter	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Output parameter	NA
Return value	Timer frequency division value
Required preconditions	NA
Called functions	NA

**Example:**

```
uint32_t div_value;
div_value = tmr_div_value_get(TMR1);
```

## 5.19.14 tmr\_output\_channel\_config function

The table below describes the function tmr\_output\_channel\_config.

**Table 402. tmr\_output\_channel\_config function**

Name	Description
Function name	tmr_output_channel_config
Function prototype	void tmr_output_channel_config(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_output_config_type *tmr_output_struct);
Function description	Configure TMR output channels
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: timer channel. Refer to <a href="#">The tmr_channel</a> is used to select a TMR channel. for details.
Input parameter 3	tmr_output_struct: tmr_output_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### tmr\_channel

The tmr\_channel is used to select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: Channel 1

TMR\_SELECT\_CHANNEL\_2: Channel 2

TMR\_SELECT\_CHANNEL\_3: Channel 3

TMR\_SELECT\_CHANNEL\_4: Channel 4

### tmr\_output\_config\_type structure

tmr\_output\_config\_type is defined in the at32f403\_tmr.h:

typedef struct

```
{
    tmr_output_control_mode_type      oc_mode;
    confirm_state                     oc_idle_state;
    confirm_state                     occ_idle_state;
    tmr_output_polarity_type         oc_polarity;
    tmr_output_polarity_type         occ_polarity;
    confirm_state                     oc_output_state;
    confirm_state                     occ_output_state;
}
```

} tmr\_output\_config\_type;

### oc\_mode

Set output channel mode, that is to configure channel original signals (CxORAW).

TMR\_OUTPUT\_CONTROL\_OFF: Disconnect channel output (CxOUT) from CxORAW

TMR\_OUTPUT\_CONTROL\_HIGH: CxORAW high

TMR\_OUTPUT\_CONTROL\_LOW: CxORAW low

TMR\_OUTPUT\_CONTROL\_SWITCH: Switch CxORAW level

TMR\_OUTPUT\_CONTROL\_FORCE\_LOW: Forced CxORAW low

TMR\_OUTPUT\_CONTROL\_FORCE\_HIGH: Forced CxORAW high

TMR\_OUTPUT\_CONTROL\_PWM\_MODE\_A: PWM A mode

TMR\_OUTPUT\_CONTROL\_PWM\_MODE\_B: PWM B mode

#### **oc\_idle\_state**

Set output channel idle state.

FALSE: Output channel idle state is 0

TRUE: Output channel idle state is 1

#### **occ\_idle\_state**

Set complementary output channel idle state.

FALSE: Complementary output channel idle state is 0

TRUE: Complementary output channel idle state is 1

#### **oc\_polarity**

Set the polarity of output channels.

TMR\_OUTPUT\_ACTIVE\_HIGH: Active high

TMR\_OUTPUT\_ACTIVE\_LOW: Active low

#### **occ\_polarity**

Set the polarity of complementary output channels.

TMR\_OUTPUT\_ACTIVE\_HIGH: Active high

TMR\_OUTPUT\_ACTIVE\_LOW: Active low

#### **oc\_output\_state**

Set the state of output channels.

FALSE: Output channel OFF

TRUE: Output channel ON

#### **occ\_output\_state**

Set the state of complementary output channels.

FALSE: Complementary output channel OFF

TRUE: Complementary output channel ON

#### **Example:**

```
tmr_output_config_type tmr_output_struct;
tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_OFF;
tmr_output_struct.oc_output_state = TRUE;
tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.oc_idle_state = TRUE;
tmr_output_struct.occ_output_state = TRUE;
tmr_output_struct.occ_polarity = TMR_OUTPUT_ACTIVE_HIGH;
tmr_output_struct.occ_idle_state = TRUE;
tmr_output_channel_config(TMR1, TMR_SELECT_CHANNEL_1, &tmr_output_struct);
```

## 5.19.15 tmr\_output\_channel\_mode\_select function

The table below describes the function tmr\_output\_channel\_mode\_select.

**Table 403. tmr\_output\_channel\_mode\_select function**

Name	Description
Function name	tmr_output_channel_mode_select
Function prototype	void tmr_output_channel_mode_select(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_output_control_mode_type oc_mode);
Function description	Select TMR output channel mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: Refer to <a href="#">The tmr_channel</a> is used to select a TMR channel. for details.
Input parameter 3	oc_mode: refer to <a href="#">oc_mode</a> for details
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### tmr\_channel

The tmr\_channel is used to select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: Channel 1

TMR\_SELECT\_CHANNEL\_2: Channel 2

TMR\_SELECT\_CHANNEL\_3: Channel 3

TMR\_SELECT\_CHANNEL\_4: Channel 4

### oc\_mode

Set output channel mode, that is to configure channel original signals (CxORAW).

TMR\_OUTPUT\_CONTROL\_OFF: Disconnect channel output (CxOUT) from CxORAW

TMR\_OUTPUT\_CONTROL\_HIGH: CxORAW high

TMR\_OUTPUT\_CONTROL\_LOW: CxORAW low

TMR\_OUTPUT\_CONTROL\_SWITCH: Switch CxORAW level

TMR\_OUTPUT\_CONTROL\_FORCE\_LOW: Forced CxORAW low

TMR\_OUTPUT\_CONTROL\_FORCE\_HIGH: Forced CxORAW high

TMR\_OUTPUT\_CONTROL\_PWM\_MODE\_A: PWM A mode

TMR\_OUTPUT\_CONTROL\_PWM\_MODE\_B: PWM B mode

### Example:

```
tmr_output_channel_mode_select(TMR1, TMR_SELECT_CHANNEL_1, TMR_OUTPUT_CONTROL_SWITCH);
```

## 5.19.16 tmr\_period\_value\_set function

The table below describes the function tmr\_period\_value\_set.

**Table 404. tmr\_period\_value\_set function**

Name	Description
Function name	tmr_period_value_set
Function prototype	void tmr_period_value_set(tmr_type *tmr_x, uint32_t tmr_pr_value);
Function description	Set TMR period value
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_pr_value: timer period value. For 16-bit timer, it can be 0x0000~0xFFFF; for 32-bit timer, it can be 0x0000_0000~0xFFFF_FFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
tmr_period_value_set(TMR1, 0xFFFF);
```

## 5.19.17 tmr\_period\_value\_get function

The table below describes the function tmr\_period\_value\_get.

**Table 405. tmr\_period\_value\_get function**

Name	Description
Function name	tmr_period_value_get
Function prototype	uint32_t tmr_period_value_get(tmr_type *tmr_x);
Function description	Get TMR period value
Input parameter	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Output parameter	NA
Return value	Timer period value
Required preconditions	NA
Called functions	NA

**Example:**

```
uint32_t pr_value;  
pr_value = tmr_period_value_get(TMR1);
```

## 5.19.18 tmr\_channel\_value\_set function

The table below describes the function tmr\_channel\_value\_set.

**Table 406. tmr\_channel\_value\_set function**

Name	Description
Function name	tmr_channel_value_set
Function prototype	void tmr_channel_value_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, uint32_t tmr_channel_value);
Function description	Set TMR channel value
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: timer channel. Refer to <a href="#">The tmr_channel</a> is used to select a TMR channel. for details.
Input parameter 3	tmr_channel_value: timer channel value. For 16-bit timer, it can be 0x0000~0xFFFF; for 32-bit timer, it can be 0x0000_0000~0xFFFF_FFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### tmr\_channel

The tmr\_channel is used to select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: Channel 1

TMR\_SELECT\_CHANNEL\_2: Channel 2

TMR\_SELECT\_CHANNEL\_3: Channel 3

TMR\_SELECT\_CHANNEL\_4: Channel 4

### Example:

```
tmr_channel_value_set(TMR1, TMR_SELECT_CHANNEL_1, 0xFFFF);
```

## 5.19.19 tmr\_channel\_value\_get function

The table below describes the function tmr\_channel\_value\_get.

**Table 407. tmr\_channel\_value\_get function**

Name	Description
Function name	tmr_channel_value_get
Function prototype	uint32_t tmr_channel_value_get(tmr_type *tmr_x, tmr_channel_select_type tmr_channel);
Function description	Get TMR channel value
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: timer channel. Refer to <a href="#">The tmr_channel</a> is used to select a TMR channel. for details.
Output parameter	Timer channel value
Return value	NA
Required preconditions	NA
Called functions	NA

**tmr\_channel**

The tmr\_channel is used to select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: Channel 1

TMR\_SELECT\_CHANNEL\_2: Channel 2

TMR\_SELECT\_CHANNEL\_3: Channel 3

TMR\_SELECT\_CHANNEL\_4: Channel 4

**Example:**

```
uint32_t ch_value;  
ch_value = tmr_channel_value_get(TMR1, TMR_SELECT_CHANNEL_1);
```

## 5.19.20 tmr\_period\_buffer\_enable function

The table below describes the function tmr\_period\_buffer\_enable.

**Table 408. tmr\_period\_buffer\_enable function**

Name	Description
Function name	tmr_period_buffer_enable
Function prototype	void tmr_period_buffer_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR period buffer
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	new_state: indicates the status of period buffer. It can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
tmr_period_buffer_enable(TMR1, TRUE);
```

## 5.19.21 tmr\_output\_channel\_buffer\_enable function

The table below describes the function tmr\_output\_channel\_buffer\_enable.

**Table 409. tmr\_output\_channel\_buffer\_enable function**

Name	Description
Function name	tmr_output_channel_buffer_enable
Function prototype	void tmr_output_channel_buffer_enable(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Enable or disable TMR output channel buffer
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: timer channel.
Input parameter 3	new_state: indicates the status of output channel buffer. It can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### tmr\_channel

The tmr\_channel is used to select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: Channel 1

TMR\_SELECT\_CHANNEL\_2: Channel 2

TMR\_SELECT\_CHANNEL\_3: Channel 3

TMR\_SELECT\_CHANNEL\_4: Channel 4

### Example:

```
tmr_output_channel_buffer_enable(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

## 5.19.22 tmr\_output\_channel\_immediately\_set function

The table below describes the function tmr\_output\_channel\_immediately\_set.

**Table 410. tmr\_output\_channel\_immediately\_set function**

Name	Description
Function name	tmr_output_channel_immediately_set
Function prototype	void tmr_output_channel_immediately_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Enable TMR output channel immediately
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of output channel enable. This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA

Name	Description
Required preconditions	NA
Called functions	NA

**tmr\_channel**

The tmr\_channel is used to select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: Channel 1

TMR\_SELECT\_CHANNEL\_2: Channel 2

TMR\_SELECT\_CHANNEL\_3: Channel 3

TMR\_SELECT\_CHANNEL\_4: Channel 4

**Example:**

```
tmr_output_channel_immediately_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

## 5.19.23 tmr\_output\_channel\_switch\_set function

The table below describes the function tmr\_output\_channel\_switch\_set.

**Table 411. tmr\_output\_channel\_switch\_set function**

Name	Description
Function name	tmr_output_channel_switch_set
Function prototype	void tmr_output_channel_switch_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Set TMR output channel switch
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: timer channel. Refer to <a href="#">The tmr_channel</a> is used to select a TMR channel. for details.
Input parameter 3	new_state: indicates the status of output channel swith. This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**tmr\_channel**

The tmr\_channel is used to select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: Channel 1

TMR\_SELECT\_CHANNEL\_2: Channel 2

TMR\_SELECT\_CHANNEL\_3: Channel 3

TMR\_SELECT\_CHANNEL\_4: Channel 4

**Example:**

```
tmr_output_channel_switch_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

## 5.19.24 tmr\_one\_cycle\_mode\_enable function

The table below describes the function tmr\_one\_cycle\_mode\_enable.

**Table 412. tmr\_one\_cycle\_mode\_enable function**

Name	Description
Function name	tmr_one_cycle_mode_enable
Function prototype	void tmr_one_cycle_mode_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR one-cycle mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	new_state: indicates the status of one-cycle mode. This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
tmr_one_cycle_mode_enable(TMR1, TRUE);
```

## 5.19.25 tmr\_32\_bit\_function\_enable function

The table below describes the function tmr\_32\_bit\_function\_enable.

**Table 413. tmr\_32\_bit\_function\_enable function**

Name	Description
Function name	tmr_32_bit_function_enable
Function prototype	void tmr_32_bit_function_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR 32-bit feature (plus mode)
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR2, TMR5
Input parameter 2	new_state: the status of 32-bit mode This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
tmr_32_bit_function_enable(TMR2, TRUE);
```

## 5.19.26 tmr\_overflow\_request\_source\_set function

The table below describes the function tmr\_overflow\_request\_source\_set.

Table 414. tmr\_overflow\_request\_source\_set function

Name	Description
Function name	tmr_overflow_request_source_set
Function prototype	void tmr_overflow_request_source_set(tmr_type *tmr_x, confirm_state new_state);
Function description	Select TMR overflow event sources
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	new_state: indicates the overflow event source.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### new\_state

Select an overflow event source.

FALSE: Counter overflow, OVFSWTR being set, overflow event from slave timer controller

TRUE: Counter overflow only.

### Example:

```
tmr_overflow_request_source_set(TMR1, TRUE);
```

## 5.19.27 tmr\_overflow\_event\_disable function

The table below describes the function tmr\_overflow\_event\_disable.

**Table 415. tmr\_overflow\_event\_disable function**

Name	Description
Function name	tmr_overflow_event_disable
Function prototype	void tmr_overflow_event_disable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR overflow event generation
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	new_state: indicates the status of overflow event generation.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### **new\_state**

Select the status of overflow event generation.

FALSE: Enable overflow event generation, which can be generated from the following:

- Counter overflow
- Set OVFSWTR=1
- Overflow event from slave timer controller

TRUE: Disable overflow event generation

### **Example:**

```
tmr_overflow_event_disable(TMR1, TRUE);
```

## 5.19.28 tmr\_input\_channel\_init function

The table below describes the function tmr\_input\_channel\_init.

**Table 416. tmr\_input\_channel\_init function**

Name	Description
Function name	tmr_input_channel_init
Function prototype	void tmr_input_channel_init(tmr_type *tmr_x, tmr_input_config_type *input_struct, tmr_channel_input_divider_type divider_factor);
Function description	Initialize TMR input channels
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	input_struct: tmr_input_config_type pointer
Input parameter 3	divider_factor: input channel frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### **tmr\_input\_config\_type structure**

tmr\_input\_config\_type is defined in the at32f403\_tmr.h:

```
typedef struct
{
    tmr_channel_select_type           input_channel_select;
    tmr_input_polarity_type          input_polarity_select;
    tmr_input_direction_mapped_type   input_mapped_select;
    uint8_t                           input_filter_value;
} tmr_input_config_type;
```

### **input\_channel\_select**

Select a TMR input channel.

TMR\_SELECT\_CHANNEL\_1: Channel 1

TMR\_SELECT\_CHANNEL\_2: Channel 2

TMR\_SELECT\_CHANNEL\_3: Channel 3

TMR\_SELECT\_CHANNEL\_4: Channel 4

### **input\_polarity\_select**

Select the polarity of input channels.

TMR\_INPUT\_RISING\_EDGE: Rising edge

TMR\_INPUT\_FALLING\_EDGE: Falling edge

TMR\_INPUT\_BOTH\_EDGE: Both edges (Rising edge and Falling edge)

### **input\_mapped\_select**

Select input channel mapping.

TMR\_CC\_CHANNEL\_MAPPED\_DIRECT:

TMR input channel 1,2,3 and 4 is linked to C1IRAW, C2IRAW, C3IRAW and C4IRAW respectively.

TMR\_CC\_CHANNEL\_MAPPED\_INDIRECT:

TMR input channel 1,2,3 and 4 is linked to C2IRAW, C1IRAW, C4IRAW and C3IRAW respectively.

TMR\_CC\_CHANNEL\_MAPPED\_STI:

TMR input channel is mapped on STI

### **input\_filter\_value**

Select an input channel filter value, between 0x00~0x0F

### **divider\_factor**

Select input channel frequency division factor.

TMR\_CHANNEL\_INPUT\_DIV\_1: Divided by 1

TMR\_CHANNEL\_INPUT\_DIV\_2: Divided by 2

TMR\_CHANNEL\_INPUT\_DIV\_4: Divided by 4

TMR\_CHANNEL\_INPUT\_DIV\_8: Divided by 8

### **Example:**

```
tmr_input_config_type tmr_input_config_struct;
tmr_input_config_struct.input_channel_select = TMR_SELECT_CHANNEL_2;
tmr_input_config_struct.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
tmr_input_config_struct.input_polarity_select = TMR_INPUT_RISING_EDGE;
tmr_input_config_struct.input_filter_value = 0x00;
tmr_input_channel_init(TMR1, &tmr_input_config_struct, TMR_CHANNEL_INPUT_DIV_1);
```

## 5.19.29 tmr\_channel\_enable function

The table below describes the function tmr\_channel\_enable.

**Table 417. tmr\_channel\_enable function**

Name	Description
Function name	tmr_channel_enable
Function prototype	void tmr_channel_enable(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, confirm_state new_state);
Function description	Enable or disable TMR channels
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: timer channel
Input parameter 3	new_state: indicates the status of timer channels. This parameter can be Enable (TRUE) or Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### tmr\_channel

Select a TMR channel.

- TMR\_SELECT\_CHANNEL\_1: Timer channel 1
- TMR\_SELECT\_CHANNEL\_1C: Complementary channel 1
- TMR\_SELECT\_CHANNEL\_2: Timer channel 2
- TMR\_SELECT\_CHANNEL\_2C: Complementary channel 2
- TMR\_SELECT\_CHANNEL\_3: Timer channel 3
- TMR\_SELECT\_CHANNEL\_3C: Complementary channel 3
- TMR\_SELECT\_CHANNEL\_4: Timer channel 4

### Example:

```
tmr_channel_enable(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

### 5.19.30 tmr\_input\_channel\_filter\_set function

The table below describes the function tmr\_input\_channel\_filter\_set.

**Table 418. tmr\_input\_channel\_filter\_set function**

Name	Description
Function name	tmr_input_channel_filter_set
Function prototype	void tmr_input_channel_filter_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, uint16_t filter_value);
Function description	Set TMR input channel filter
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: timer channel. Refer to <a href="#">The tmr_channel</a> is used to select a TMR channel. for details.
Input parameter 3	filter_value: set channel filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### tmr\_channel

Select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: Timer channel 1

TMR\_SELECT\_CHANNEL\_2: Timer channel 2

TMR\_SELECT\_CHANNEL\_3: Timer channel 3

TMR\_SELECT\_CHANNEL\_4: Timer channel 4

#### Example:

```
tmr_input_channel_filter_set(TMR1, TMR_SELECT_CHANNEL_1, 0x0F);
```

### 5.19.31 tmr\_pwm\_input\_config function

The table below describes the function tmr\_pwm\_input\_config.

**Table 419. tmr\_pwm\_input\_config function**

Name	Description
Function name	tmr_pwm_input_config
Function prototype	void tmr_pwm_input_config(tmr_type *tmr_x, tmr_input_config_type *input_struct, tmr_channel_input_divider_type divider_factor);
Function description	Configure TMR pwm input
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	input_struct: tmr_input_config_type pointer
Input parameter 3	divider_factor: input channel frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA

Name	Description
Called functions	NA

**input\_struct**

Point to the tmr\_input\_config\_type, see [tmr\\_input\\_config\\_type](#) for details.

**divider\_factor**

Select input channel frequency division factor

- TMR\_CHANNEL\_INPUT\_DIV\_1: Divided by 1
- TMR\_CHANNEL\_INPUT\_DIV\_2: Divided by 2
- TMR\_CHANNEL\_INPUT\_DIV\_4: Divided by 4
- TMR\_CHANNEL\_INPUT\_DIV\_8: Divided by 8

**Example:**

```
tmr_input_config_type tmr_ic_init_structure;
tmr_ic_init_structure.input_filter_value = 0;
tmr_ic_init_structure.input_channel_select = TMR_SELECT_CHANNEL_2;
tmr_ic_init_structure.input_mapped_select = TMR_CC_CHANNEL_MAPPED_DIRECT;
tmr_ic_init_structure.input_polarity_select = TMR_INPUT_RISING_EDGE;
tmr_pwm_input_config(TMR1, &tmr_ic_init_structure, TMR_CHANNEL_INPUT_DIV_1);
```

### 5.19.32 tmr\_channel1\_input\_select function

The table below describes the function tmr\_channel1\_input\_select.

**Table 420. tmr\_channel1\_input\_select function**

Name	Description
Function name	tmr_channel1_input_select
Function prototype	void tmr_channel1_input_select(tmr_type *tmr_x, tmr_channel1_input_connected_type ch1_connect);
Function description	Select TMR channel 1 input
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR15
Input parameter 2	ch1_connect: channel 1 input selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**ch1\_connect**

Select channel 1 input.

TMR\_CHANNEL1\_CONNECTED\_C1IRAW: CH1 pin is connected to C1IRAW

TMR\_CHANNEL1\_2\_3\_CONNECTED\_C1IRAW\_XOR: Connect the XOR results of CH1, CH2 and CH3 pins to C1IRAW

**Example:**

```
tmr_channel1_input_select(TMR1, TMR_CHANNEL1_2_3_CONNECTED_C1IRAW_XOR);
```

### 5.19.33 tmr\_input\_channel\_divider\_set function

The table below describes the function tmr\_input\_channel\_divider\_set.

**Table 421. tmr\_input\_channel\_divider\_set function**

Name	Description
Function name	tmr_input_channel_divider_set
Function prototype	void tmr_input_channel_divider_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_channel_input_divider_type divider_factor);
Function description	Set TMR input channel divider
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: timer channel. Refer to <a href="#">The tmr_channel</a> is used to select a TMR channel. for details.
Input parameter 3	divider_factor: input channel frequency division factor
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### tmr\_channel

Select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: Timer channel 1

TMR\_SELECT\_CHANNEL\_2: Timer channel 2

TMR\_SELECT\_CHANNEL\_3: Timer channel 3

TMR\_SELECT\_CHANNEL\_4: Timer channel 4

#### divider\_factor

Select input channel frequency division factor

TMR\_CHANNEL\_INPUT\_DIV\_1: Divided by 1

TMR\_CHANNEL\_INPUT\_DIV\_2: Divided by 2

TMR\_CHANNEL\_INPUT\_DIV\_4: Divided by 4

TMR\_CHANNEL\_INPUT\_DIV\_8: Divided by 8

#### Example:

```
tmr_input_channel_divider_set(TMR1, TMR_SELECT_CHANNEL_1, TMR_CHANNEL_INPUT_DIV_2);
```

### 5.19.34 tmr\_primary\_mode\_select function

The table below describes the function tmr\_primary\_mode\_select.

**Table 422. tmr\_primary\_mode\_select function**

Name	Description
Function name	tmr_primary_mode_select
Function prototype	void tmr_primary_mode_select(tmr_type *tmr_x, tmr_primary_select_type primary_mode);
Function description	Select TMR master mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR15

Name	Description
Input parameter 2	primary_mode: master mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**primary\_mode**

Select master mode, that is, master timer output signal selection.

TMR_PRIMARY_SEL_RESET:	Reset
TMR_PRIMARY_SEL_ENABLE:	Enable
TMR_PRIMARY_SEL_OVERFLOW:	Overflow
TMR_PRIMARY_SEL_COMPARE:	Compare
TMR_PRIMARY_SEL_C1ORAW:	C1ORAW
TMR_PRIMARY_SEL_C2ORAW:	C2ORAW
TMR_PRIMARY_SEL_C3ORAW:	C3ORAW
TMR_PRIMARY_SEL_C4ORAW:	C4ORAW

**Example:**

```
tmr_primary_mode_select(TMR1, TMR_PRIMARY_SEL_RESET);
```

### 5.19.35 tmr\_sub\_mode\_select function

The table below describes the function tmr\_sub\_mode\_select.

**Table 423. tmr\_sub\_mode\_select function**

Name	Description
Function name	tmr_sub_mode_select
Function prototype	void tmr_sub_mode_select(tmr_type *tmr_x, tmr_sub_mode_select_type sub_mode);
Function description	Select TMR slave timer mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR12, TMR15
Input parameter 2	sub_mode: slave timer mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**primary\_mode****primary\_mode**

Select slave timer modes.

TMR_SUB_MODE_DIABLE:	Disable
TMR_SUB_ENCODER_MODE_A:	Encoder mode A
TMR_SUB_ENCODER_MODE_B:	Encoder mode B
TMR_SUB_ENCODER_MODE_C:	Encoder mode C
TMR_SUB_RESET_MODE:	Reset
TMR_SUB_HANG_MODE:	Suspend
TMR_SUB_TRIGGER_MODE:	Trigger
TMR_SUB_EXTERNAL_CLOCK_MODE_A:	External clock A

**Example:**

```
tmr_sub_mode_select(TMR1, TMR_SUB_HANG_MODE);
```

**5.19.36 tmr\_channel\_dma\_select function**

The table below describes the function tmr\_channel\_dma\_select.

**Table 424. tmr\_channel\_dma\_select function**

Name	Description
Function name	tmr_channel_dma_select
Function prototype	void tmr_channel_dma_select(tmr_type *tmr_x, tmr_dma_request_source_type cc_dma_select);
Function description	Select TMR channel DMA request source
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR12, TMR15
Input parameter 2	cc_dma_select: TMR channel DMA request source
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**cc\_dma\_select**

Select DMA request source for TMR channels.

TMR\_DMA\_REQUEST\_BY\_CHANNEL: DMA request upon a channel event (CxIF = 1)

TMR\_DMA\_REQUEST\_BY\_OVERFLOW: DMA request upon an overflow event (OVFIF = 1)

**Example:**

```
tmr_channel_dma_select(TMR1, TMR_DMA_REQUEST_BY_OVERFLOW);
```

**5.19.37 tmr\_hall\_select function**

The table below describes the function tmr\_hall\_select.

**Table 425. tmr\_hall\_select function**

Name	Description
Function name	tmr_hall_select
Function prototype	void tmr_hall_select(tmr_type *tmr_x, confirm_state new_state);
Function description	Select TMR hall mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR8, TMR15
Input parameter 2	new_state: indicates the status of TMR hall mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**new\_state**

Select the status of TMR hall mode in order to refresh channel control bit.

FALSE: Refresh channel control bit through HALL

TRUE: Refresh channel control bit through HALL or the rising edge of TRGIN

**Example:**

```
tmr_hall_select(TMR1, TRUE);
```

### 5.19.38 tmr\_channel\_buffer\_enable function

The table below describes the function tmr\_channel\_buffer\_enable.

**Table 426. tmr\_channel\_buffer\_enable function**

Name	Description
Function name	tmr_channel_buffer_enable
Function prototype	void tmr_channel_buffer_enable(tmr_type *tmr_x, confirm_state new_state);
Function description	Enable or disable TMR channel buffer
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR8, TMR15
Input parameter 2	new_state: indicates the status of TMR channel buffer. This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
tmr_channel_buffer_enable(TMR1, TRUE);
```

### 5.19.39 tmr\_trigger\_input\_select function

The table below describes the function tmr\_trigger\_input\_select.

**Table 427. tmr\_trigger\_input\_select function**

Name	Description
Function name	tmr_trigger_input_select
Function prototype	void tmr_trigger_input_select(tmr_type *tmr_x, sub_tmr_input_sel_type trigger_select);
Function description	Select TMR slave timer trigger input
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR12, TMR15
Input parameter 2	trigger_select: select TMR slave timer trigger input
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**trigger\_select**

Select TMR slave timer trigger input.

- TMR\_SUB\_INPUT\_SEL\_IS: Internal input 0
- TMR\_SUB\_INPUT\_SEL\_IS1: Internal input 1
- TMR\_SUB\_INPUT\_SEL\_IS2: Internal input 2
- TMR\_SUB\_INPUT\_SEL\_IS3: Internal input 3
- TMR\_SUB\_INPUT\_SEL\_C1INC: C1IRAW input detection
- TMR\_SUB\_INPUT\_SEL\_C1DF1: Filter input channel 1
- TMR\_SUB\_INPUT\_SEL\_C2DF2: Filter input channel 2
- TMR\_SUB\_INPUT\_SEL\_EXTIN: External input channel EXT

**Example:**

```
tmr_trigger_input_select(TMR1, TMR_SUB_INPUT_SEL_IS0);
```

### 5.19.40 tmr\_sub\_sync\_mode\_set function

The table below describes the function tmr\_sub\_sync\_mode\_set.

**Table 428. tmr\_sub\_sync\_mode\_set function**

Name	Description
Function name	tmr_sub_sync_mode_set
Function prototype	void tmr_sub_sync_mode_set(tmr_type *tmr_x, confirm_state new_state);
Function description	Set TMR slave timer synchronization mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR12, TMR15
Input parameter 2	new_state: indicates the status of TMR slave timer synchronization mode This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

tmr_sub_sync_mode_set(TMR1, TRUE);
------------------------------------

### 5.19.41 tmr\_dma\_request\_enable function

The table below describes the function tmr\_dma\_request\_enable.

**Table 429. tmr\_dma\_request\_enable function**

Name	Description
Function name	tmr_dma_request_enable
Function prototype	void tmr_dma_request_enable(tmr_type *tmr_x, tmr_dma_request_type dma_request, confirm_state new_state);
Function description	Enable or disable TMR DMA request
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	dma_request: DMA request
Input parameter 3	new_state: indicates the status of DMA request. This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**dma\_request**

Select a DMA request.

TMR\_OVERFLOW\_DMA\_REQUEST: Overflow event DMA request

TMR\_C1\_DMA\_REQUEST: Channel 1 DMA request

TMR\_C2\_DMA\_REQUEST: Channel 2 DMA request

TMR\_C3\_DMA\_REQUEST: Channel 3 DMA request

TMR\_C4\_DMA\_REQUEST: Channel 4 DMA request

TMR\_HALL\_DMA\_REQUEST: HALL event DMA request  
 TMR\_TRIGGER\_DMA\_REQUEST: Trigger event DMA request

**Example:**

```
tmr_dma_request_enable(TMR1, TMR_OVERFLOW_DMA_REQUEST, TRUE);
```

### 5.19.42 tmr\_interrupt\_enable function

The table below describes the function tmr\_interrupt\_enable.

**Table 430. tmr\_interrupt\_enable function**

Name	Description
Function name	tmr_interrupt_enable
Function prototype	void tmr_interrupt_enable(tmr_type *tmr_x, uint32_t tmr_interrupt, confirm_state new_state);
Function description	Enable or disable TMR interrupts
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_interrupt: TMR interrupts
Input parameter 3	new_state: indicates the status of TMR interrupts. This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### tmr\_interrupt

Select a TMR interrupt.

TMR\_OVF\_INT: Overflow event interrupt  
 TMR\_C1\_INT: Channel 1 event interrupt  
 TMR\_C2\_INT: Channel 2 event interrupt  
 TMR\_C3\_INT: Channel 3 event interrupt  
 TMR\_C4\_INT: Channel 4 event interrupt  
 TMR\_HALL\_INT: HALL event interrupt  
 TMR\_TRIGGER\_INT: Trigger event interrupt  
 TMR\_BRK\_INT: Break event interrupt

**Example:**

```
tmr_interrupt_enable(TMR1, TMR_OVF_INT, TRUE);
```

## 5.19.43 tmr\_interrupt\_flag\_get function

The table below describes the function tmr\_interrupt\_flag\_get.

**Table 431. tmr\_interrupt\_flag\_get function**

Name	Description
Function name	tmr_interrupt_flag_get
Function prototype	flag_status tmr_interrupt_flag_get (tmr_type *tmr_x, uint32_t tmr_flag);
Function description	Get the interrupt flag status
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_flag: Interrupt flag selection Refer to the “tmr_flag” description below for details.
Output parameter	NA
Return value	flag_status: indicates the status of flags Return SET or RESET.
Required preconditions	NA
Called functions	NA

### tmr\_flag

This is used for flag selection, including:

TMR_OVF_FLAG:	Overflow interrupt flag
TMR_C1_FLAG:	Channel 1 interrupt flag
TMR_C2_FLAG:	Channel 2 interrupt flag
TMR_C3_FLAG:	Channel 3 interrupt flag
TMR_C4_FLAG:	Channel 4 interrupt flag
TMR_HALL_FLAG:	HALL interrupt flag
TMR_TRIGGER_FLAG:	Trigger interrupt flag
TMR_BRK_FLAG:	Break interrupt flag

### Example

```
if(tmr_interrupt_flag_get (TMR1, TMR_OVF_FLAG) != RESET)
```

## 5.19.44 tmr\_flag\_get function

The table below describes the function tmr\_flag\_get.

Table 432. tmr\_flag\_get function

Name	Description
Function name	tmr_flag_get
Function prototype	flag_status tmr_flag_get(tmr_type *tmr_x, uint32_t tmr_flag);
Function description	Get flag status
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_flag: Flag selection Refer to the “tmr_flag” description below for details.
Output parameter	NA
Return value	flag_status: indicates the status of flags Return SET or RESET.
Required preconditions	NA
Called functions	NA

### tmr\_flag

This is used for flag selection, including:

TMR_OVF_FLAG:	Overflow interrupt flag
TMR_C1_FLAG:	Channel 1 interrupt flag
TMR_C2_FLAG:	Channel 2 interrupt flag
TMR_C3_FLAG:	Channel 3 interrupt flag
TMR_C4_FLAG:	Channel 4 interrupt flag
TMR_HALL_FLAG:	HALL interrupt flag
TMR_TRIGGER_FLAG:	Trigger interrupt flag
TMR_BRK_FLAG:	Break interrupt flag
TMR_C1_RECAPTURE_FLAG:	Channel 1 recapture flag
TMR_C2_RECAPTURE_FLAG:	Channel 2 recapture flag
TMR_C3_RECAPTURE_FLAG:	Channel 3 recapture flag
TMR_C4_RECAPTURE_FLAG:	Channel 4 recapture flag

### Example:

```
if(tmr_flag_get(TMR1, TMR_OVF_FLAG) != RESET)
```

## 5.19.45 tmr\_flag\_clear function

The table below describes the function tmr\_flag\_clear.

**Table 433. tmr\_flag\_clear function**

Name	Description
Function name	tmr_flag_clear
Function prototype	void tmr_flag_clear(tmr_type *tmr_x, uint32_t tmr_flag);
Function description	Clear flag
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_flag: flag selection Refer to <a href="#">tmr_flag</a> for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
tmr_flag_clear(TMR1, TMR_OVF_FLAG);
```

## 5.19.46 tmr\_event\_sw\_trigger function

The table below describes the function tmr\_event\_sw\_trigger

**Table 434. tmr\_event\_sw\_trigger function**

Name	Description
Function name	tmr_event_sw_trigger
Function prototype	void tmr_event_sw_trigger(tmr_type *tmr_x, tmr_event_trigger_type tmr_event);
Function description	Software triggers TMR events
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR6, TMR7, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_event: select a TMR event
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**tmr\_event**

Set TMR events triggered by software.

TMR_OVERFLOW_SWTRIG:	Overflow event
TMR_C1_SWTRIG:	Channel 1 event
TMR_C2_SWTRIG:	Channel 2 event
TMR_C3_SWTRIG:	Channel 3 event
TMR_C4_SWTRIG:	Channel 4 event
TMR_HALL_SWTRIG:	HALL event
TMR_TRIGGER_SWTRIG:	Trigger event
TMR_BRK_SWTRIG:	Break event

**Example:**

<code>tmr_event_sw_trigger(TMR1, TMR_OVERFLOW_SWTRIG);</code>
---

### 5.19.47 tmr\_output\_enable function

The table below describes the function tmr\_output\_enable.

**Table 435. tmr\_output\_enable function**

Name	Description
Function name	<code>tmr_output_enable</code>
Function prototype	<code>void tmr_output_enable(tmr_type *tmr_x, confirm_state new_state);</code>
Function description	Enable or disable TMR output
Input parameter 1	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR8, TMR15
Input parameter 2	<code>new_state</code> : TMR output status This parameter can be Enable (TRUE) or Disable (FALSE).
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

<code>tmr_output_enable(TMR1, TRUE);</code>
---

### 5.19.48 tmr\_internal\_clock\_set function

The table below describes the function tmr\_internal\_clock\_set.

**Table 436. tmr\_internal\_clock\_set function**

Name	Description
Function name	<code>tmr_internal_clock_set</code>
Function prototype	<code>void tmr_internal_clock_set(tmr_type *tmr_x);</code>
Function description	Set TMR internal clock
Input parameter	<code>tmr_x</code> : indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR12, TMR15
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

<code>tmr_internal_clock_set(TMR1);</code>
--

## 5.19.49 tmr\_output\_channel\_polarity\_set function

The table below describes the function tmr\_output\_channel\_polarity\_set.

**Table 437. tmr\_output\_channel\_polarity\_set function**

Name	Description
Function name	tmr_output_channel_polarity_set
Function prototype	void tmr_output_channel_polarity_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_polarity_active_type oc_polarity);
Function description	Set TMR output channel polarity
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: Timer channel. Refer to <a href="#">tmr_flag</a> for details.
Input parameter 3	oc_polarity: output channel polarity
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### tmr\_channel

Select a TMR channel.

TMR_SELECT_CHANNEL_1:	Timer channel 1
TMR_SELECT_CHANNEL_1C:	Timer complementary channel 1
TMR_SELECT_CHANNEL_2:	Timer channel 2
TMR_SELECT_CHANNEL_2C:	Timer complementary channel 2
TMR_SELECT_CHANNEL_3:	Timer channel 3
TMR_SELECT_CHANNEL_3C:	Timer complementary channel 3
TMR_SELECT_CHANNEL_4:	Timer channel 4

### oc\_polarity

Select TMR channel polarity.

TMR\_POLARITY\_ACTIVE\_HIGH: Active high

TMR\_POLARITY\_ACTIVE\_LOW: Active low

### Example:

```
tmr_output_channel_polarity_set(TMR1, TMR_SELECT_CHANNEL_1, TMR_POLARITY_ACTIVE_HIGH);
```

## 5.19.50 tmr\_external\_clock\_config function

The table below describes the function tmr\_external\_clock\_config.

**Table 438. tmr\_external\_clock\_config function**

Name	Description
Function name	tmr_external_clock_config
Function prototype	void tmr_external_clock_config(tmr_type *tmr_x, tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type es_polarity, uint16_t es_filter);
Function description	Configure TMR external clock
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR15

Name	Description
Input parameter 2	es_divide: external signal frequency division factor
Input parameter 3	es_polarity: external signal polarity
Input parameter 4	es_filter: external signal filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**es\_divide**

Set TMR external signal frequency division factor.

TMR\_ES\_FREQUENCY\_DIV\_1: Divided by 1

TMR\_ES\_FREQUENCY\_DIV\_2: Divided by 2

TMR\_ES\_FREQUENCY\_DIV\_4: Divided by 4

TMR\_ES\_FREQUENCY\_DIV\_8: Divided by 8

**es\_polarity**

Select TMR external signal polarity.

TMR\_ES\_POLARITY\_NON\_INVERTED: Hiht or risging edge

TMR\_ES\_POLARITY\_INVERTED: Low or falling edge

**Example:**

```
tmr_external_clock_config(TMR1, TMR_ES_FREQUENCY_DIV_1, TMR_ES_POLARITY_INVERTED, 0x0F);
```

## 5.19.51 tmr\_external\_clock\_mode1\_config function

The table below describes the function tmr\_external\_clock\_mode1\_config.

**Table 439. tmr\_external\_clock\_mode1\_config function**

Name	Description
Function name	tmr_external_clock_mode1_config
Function prototype	void tmr_external_clock_mode1_config(tmr_type *tmr_x, tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type es_polarity, uint16_t es_filter);
Function description	Configure TMR external clock mode 1 (corresponding to external mode A in the reference manual)
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR15
Input parameter 2	es_divide: external signal frequency division factor
Input parameter 3	es_polarity: external signal polarity
Input parameter 4	es_filter: external signal filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**es\_divide**

Set TMR external signal frequency division factor, refer to [es\\_divide](#) for details.

**es\_polarity**

Set TMR external signal polarity, refer to [es\\_polarity](#) for details.

**Example:**

```
tmr_external_clock_mode1_config(TMR1, TMR_ES_FREQUENCY_DIV_1, TMR_ES_POLARITY_INVERTED,  
0x0F);
```

## 5.19.52 tmr\_external\_clock\_mode2\_config function

The table below describes the function tmr\_external\_clock\_mode2\_config.

**Table 440. tmr\_external\_clock\_mode2\_config function**

Name	Description
Function name	tmr_external_clock_mode2_config
Function prototype	void tmr_external_clock_mode2_config(tmr_type *tmr_x, tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type es_polarity, uint16_t es_filter);
Function description	Configure TMR external clock mode 2 (corresponding to external mode B in the reference manual)
Input parameter 1	tmr_x : indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR15
Input parameter 2	es_divide: external signal frequency division factor
Input parameter 3	es_polarity: external signal polarity
Input parameter 4	es_filter: external signal filter value, 0x00~0x0F
Output parameter	NA
Return value	NA
Input parameter 2	es_divide: external signal frequency division factor
Input parameter 3	es_polarity: external signal polarity

**es\_divide**

Set TMR external signal frequency division factor, refer to [es\\_divide](#) for details.

**es\_polarity**

Set TMR external signal polarity, refer to [es\\_polarity](#)

for details.

**Example:**

```
tmr_external_clock_mode2_config(TMR1, TMR_ES_FREQUENCY_DIV_1, TMR_ES_POLARITY_INVERTED,  
0x0F);
```

### 5.19.53 tmr\_encoder\_mode\_config function

The table below describes the function tmr\_encoder\_mode\_config.

**Table 441. tmr\_encoder\_mode\_config function**

Name	Description
Function name	tmr_encoder_mode_config
Function prototype	void tmr_encoder_mode_config(tmr_type *tmr_x, tmr_encoder_mode_type encoder_mode, tmr_input_polarity_type ic1_polarity, tmr_input_polarity_type ic2_polarity);
Function description	Configure TMR encoder mode
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8,TMR15
Input parameter 2	encoder_mode: encoder mode
Input parameter 3	ic1_polarity: input channel 1 polarity
Input parameter 4	ic2_polarity: input channel 2 polarity
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### **encoder\_mode**

Select a TMR encoder mode.

TMR\_ENCODER\_MODE\_A: Encoder mode A

TMR\_ENCODER\_MODE\_B: Encoder mode B

TMR\_ENCODER\_MODE\_C: Encoder mode C

#### **ic1\_polarity**

Select TMR input channel 1 polarity.

TMR\_INPUT\_RISING\_EDGE: Rising edge

TMR\_INPUT\_FALLING\_EDGE: Falling edge

TMR\_INPUT\_BOTH\_EDGE: Both edges (Rising edge and Falling edge)

#### **ic2\_polarity**

Select TMR input channel 2 polarity.

TMR\_INPUT\_RISING\_EDGE: Rising edge

TMR\_INPUT\_FALLING\_EDGE: Falling edge

TMR\_INPUT\_BOTH\_EDGE: Both edges (Rising edge and Falling edge)

#### **Example:**

```
tmr_encoder_mode_config(TMR1, TMR_ENCODER_MODE_A, TMR_INPUT_RISING_EDGE,  
TMR_INPUT_RISING_EDGE);
```

## 5.19.54 tmr\_force\_output\_set function

The table below describes the function tmr\_force\_output\_set.

**Table 442. tmr\_force\_output\_set function**

Name	Description
Function name	tmr_force_output_set
Function prototype	void tmr_force_output_set(tmr_type *tmr_x, tmr_channel_select_type tmr_channel, tmr_force_output_type force_output);
Function description	Set TMR forced output
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR9, TMR10, TMR11, TMR12, TMR13, TMR14, TMR15
Input parameter 2	tmr_channel: timer channel. Refer to <a href="#">tmr_flag</a> for details.
Input parameter 3	force_output: forced output level
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### tmr\_channel

Select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: Timer channel 1

TMR\_SELECT\_CHANNEL\_2: Timer channel 2

TMR\_SELECT\_CHANNEL\_3: Timer channel 3

TMR\_SELECT\_CHANNEL\_4: Timer channel 4

### force\_output

Forced output level of output channels.

TMR\_FORCE\_OUTPUT\_HIGH: CxORAW forced high

TMR\_FORCE\_OUTPUT\_LOW: CxORAW forced low

### Example:

```
tmr_force_output_set(TMR1, TMR_SELECT_CHANNEL_1, TMR_FORCE_OUTPUT_HIGH);
```

## 5.19.55 tmr\_dma\_control\_config function

The table below describes the function tmr\_dma\_control\_config.

**Table 443. tmr\_dma\_control\_config function**

Name	Description
Function name	tmr_dma_control_config
Function prototype	void tmr_dma_control_config(tmr_type *tmr_x, tmr_dma_transfer_length_type dma_length, tmr_dma_address_type dma_base_address);
Function description	Configure TMR DMA control
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR2, TMR3, TMR4, TMR5, TMR8, TMR15
Input parameter 2	dma_length: DMA transfer length
Input parameter 3	dma_base_address: DMA transfer offset address
Output parameter	NA

Name	Description
Return value	NA
Required preconditions	NA
Called functions	NA

**dma\_length**

Set DAM transfer bytes, including:

TMR\_DMA\_TRANSFER\_1BYTE: 1 byte

TMR\_DMA\_TRANSFER\_2BYTES: 2 bytes

TMR\_DMA\_TRANSFER\_3BYTES: 3 bytes

...

TMR\_DMA\_TRANSFER\_17BYTES: 17 bytes

TMR\_DMA\_TRANSFER\_18BYTES: 18 bytes

**dma\_base\_address**

Set DMA transfer offset address, starting from TMR control register 1, including:

TMR\_CTRL1\_ADDRESS

TMR\_CTRL2\_ADDRESS

TMR\_STCTRL\_ADDRESS

TMR\_IDEN\_ADDRESS

TMRISTS\_ADDRESS

TMR\_SWEVT\_ADDRESS

TMR\_CM1\_ADDRESS

TMR\_CM2\_ADDRESS

TMR\_CCTRL\_ADDRESS

TMR\_CVAL\_ADDRESS

TMR\_DIV\_ADDRESS

TMR\_PR\_ADDRESS

TMR\_RPR\_ADDRESS

TMR\_C1DT\_ADDRESS

TMR\_C2DT\_ADDRESS

TMR\_C3DT\_ADDRESS

TMR\_C4DT\_ADDRESS

TMR\_BRK\_ADDRESS

TMR\_DMACTRL\_ADDRESS

**Example:**

```
tmr_dma_control_config(TMR1, TMR_DMA_TRANSFER_8BYTES, TMR_CTRL1_ADDRESS);
```

## 5.19.56 tmr\_brkdt\_config function

The table below describes the function tmr\_brkdt\_config.

Table 444. tmr\_brkdt\_config function

Name	Description
Function name	tmr_brkdt_config
Function prototype	void tmr_brkdt_config(tmr_type *tmr_x, tmr_brkdt_config_type *brkdt_struct);
Function description	Configure TMR break mode and dead time
Input parameter 1	tmr_x: indicates the selected TMR peripheral, it can be TMR1, TMR8,TMR15

Name	Description
Input parameter 2	brkdt_struct: tmr_brkdt_config_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**tmr\_brkdt\_config\_type structure**

The tmr\_brkdt\_config\_type is defined in the at32f403\_tmr.h:

typedef struct

```
{
    uint8_t                      deadtime;
    tmr_brk_polarity_type        brk_polarity;
    tmr_wp_level_type            wp_level;
    confirm_state                auto_output_enable;
    confirm_state                fcsoen_state;
    confirm_state                fcsodis_state;
    confirm_state                brk_enable;
} tmr_brkdt_config_type;
```

**deadtime**

Set dead time, between 0x00~0xFF

**brk\_polarity**

Select break input polarity

TMR\_BRK\_INPUT\_ACTIVE\_LOW: Active low

TMR\_BRK\_INPUT\_ACTIVE\_HIGH: Active high

**wp\_level**

Set write protection level.

TMR\_WP\_OFF: Write protection OFF

**TMR\_WP\_LEVEL\_3:**

Level 3 write protection, protecting the bits below:

- TMRx\_BRK: DTC, BRKEN, BRKV and AOEN
- TMRx\_CTRL2: CxIOS and CxCIOS

**TMR\_WP\_LEVEL\_2:**

Level 2 write protection, protecting the bits below in addition to level-3 protected bits:

- TMRx\_CCTRL: CxP and CxCP
- TMRx\_BRK: FCSODIS and FCSEN

**TMR\_WP\_LEVEL\_1:**

Level 1 write protection, protecting the bits below in addition to level-2 protected bits:

- TMRx\_CMx: CxOCTRL and CxOBEN

**auto\_output\_enable**

Enable auto output, Enable (TRUE) or disable (FALSE)

**fcsoen\_state**

Indicates the frozen status when main output is ON. It is used to configure the status of complementary output channels when timer is OFF and output is enabled (OEN=1).

FALSE: Disable CxOUT/CxCOUT output

TRUE: Enable CxOUT/CxCOUT output, inactive level

**fcsodis\_state**

Indicates the frozen status when main output is OFF. It is used to configure the status of complementary output channels when timer is OFF and output is disabled (OEN=0).

FALSE: Disable CxOUT/CxCOUT output

TRUE: Enable CxOUT/CxCOUT output, idle level

#### **brk\_enable**

Enable break feature, Enable (TRUE) or disable (FALSE).

#### **Example**

```
tmr_brkdt_config_type tmr_brkdt_config_struct;
tmr_brkdt_config_struct.brk_enable = TRUE;
tmr_brkdt_config_struct.auto_output_enable = TRUE;
tmr_brkdt_config_struct.deadtime = 0;
tmr_brkdt_config_struct.fcsodis_state = TRUE;
tmr_brkdt_config_struct.fcsoen_state = TRUE;
tmr_brkdt_config_struct.brk_polarity = TMR_BRK_INPUT_ACTIVE_HIGH;
tmr_brkdt_config_struct.wp_level = TMR_WP_OFF;
tmr_brkdt_config(TMR1, &tmr_brkdt_config_struct);
```

## **5.20 Universal synchronous/asynchronous receiver/transmitter (USART)**

The USART register structure usart\_type is defined in the “at32f403\_usart.h”.

```
/**
 * @brief type define usart register all
 */
typedef struct
{
    ...
} usart_type;
```

The table below gives a list of the USART registers.

**Table 445. Summary of USART registers**

Register	Description
sts	Status register
dt	Data register
baudr	Baud rate register
ctrl1	Control register 1
ctrl2	Control register 2
ctrl3	Control register 3
gdiv	Guard time and divider Control register 1

The table below gives a list of USART library functions.

**Table 446. Summary of USART library functions**

Function name	Description
usart_reset	Reset USART peripheral registers
usart_init	Set baud rate, data bits and stop bits.
usart_parity_selection_config	Parity selection

uart_enable	Enable USART peripherals
uart_transmitter_enable	Enable USART transmitter
uart_receiver_enable	Enable USART receiver
uart_clock_config	Set clock polarity and phases for synchronization
uart_clock_enable	Set clock output for synchronization
uart_interrupt_enable	Enable interrupts
uart_dma_transmitter_enable	Enable DMA transmitter
uart_dma_receiver_enable	Enable DMA receiver
uart_wakeup_id_set	Set wakeup ID
uart_wakeup_mode_set	Set wakeup mode
uart_receiver_mute_enable	Enable receiver mute mode
uart_break_bit_num_set	Set break frame length
uart_lin_mode_enable	Enable LIN mode
uart_data_transmit	Data transmit
uart_data_receive	Data receive
uart_break_send	Send break frame
uart_smartcard_guard_time_set	Set smartcard guard time
uart_irda_smartcard_division_set	Set infrared and smartcard division
uart_smartcard_mode_enable	Enable smartcard mode
uart_smartcard_nack_set	Enable smartcard NACK
uart_single_line_halfduplex_select	Enable single-wire half-duplex mode
uart_irda_mode_enable	Enable infrared mode
uart_irda_low_power_enable	Enable infrared low-power mode
uart_hardware_flow_control_set	Enable hardware flow control
uart_flag_get	Get flag
uart_flag_clear	Clear flag

### 5.20.1 usart\_reset function

The table below describes the function usart\_reset.

Table 447. usart\_reset function

Name	Description
Function name	usart_reset
Function prototype	void usart_reset(usart_type* usart_x);
Function description	Reset USART peripheral registers
Input parameter 1	usart_x: indicates the selected peripherals, it can be USART1, USART2, and USART3...
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	crm_periph_reset

**Example:**

```
/* reset usart1 */
usart_reset(USART1);
```

## 5.20.2 usart\_init function

The table below describes the function usart\_init.

Table 448. usart\_init function

Name	Description
Function name	usart_init
Function prototype	void usart_init(usart_type* usart_x, uint32_t baud_rate, usart_data_bit_num_type data_bit, usart_stop_bit_num_type stop_bit);
Function description	Set baud rate, data bits and stop bit.
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3...
Input parameter 2	baud_rate: baud rate for serial interfaces
Input parameter 3	data_bit: data bit width for serial interfaces
Input parameter 4	stop_bit: stop bit width for serial interfaces
Output parameter	NA
Return value	NA
Required preconditions	This operation can be allowed only when external low-speed clock is disabled.
Called functions	NA

### **data\_bit**

Select data bit size for serial interface communication.

USART\_DATA\_8BITS: 8-bit

USART\_DATA\_9BITS: 9-bit

### **stop\_bit**

Select stop bit size for serial interface communication.

USART\_STOP\_1\_BIT: 1 bit

USART\_STOP\_0\_5\_BIT: 0.5 bit

USART\_STOP\_2\_BIT: 2 bit

USART\_STOP\_1\_5\_BIT: 1.5 bit

### **Example:**

```
/* configure uart param */  
usart_init(USART1, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
```

### 5.20.3 usart\_parity\_selection\_config function

The table below describes the function usart\_parity\_selection\_config.

**Table 449. usart\_parity\_selection\_config function**

Name	Description
Function name	usart_parity_selection_config
Function prototype	void usart_parity_selection_config(usart_type* usart_x, usart_parity_selection_type parity);
Function description	Parity selection
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3...
Input parameter 2	Parity: parity mode for serial interface communication
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### parity

Select parity mode for serial interface communication.

USART\_PARITY\_NONE: No parity

USART\_PARITY EVEN: Even

USART\_PARITY ODD: Odd

#### Example:

```
/* config usart even parity */
usart_parity_selection_config(USART1, USART_PARITY EVEN);
```

### 5.20.4 usart\_enable function

The table below describes the function usart\_enable.

**Table 450. usart\_enable function**

Name	Description
Function name	usart_enable
Function prototype	void usart_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3...
Input parameter 2	new_state: Enable (TRUE) and disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### Example:

```
/* enable usart1 */
usart_enable(USART1, TRUE);
```

## 5.20.5 usart\_transmitter\_enable function

The table below describes the function usart\_transmitter\_enable.

**Table 451. usart\_transmitter\_enable function**

Name	Description
Function name	usart_transmitter_enable
Function prototype	void usart_transmitter_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART transmitter
Input parameter 1	usart_x: indicates the selected peripheral it can be USART1, USART2, and USART3...
Input parameter 2	new_state: Enable (TRUE) and disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable usart1 transmitter */
usart_transmitter_enable(USART1, TRUE);
```

## 5.20.6 usart\_receiver\_enable function

The table below describes the function usart\_receiver\_enable.

**Table 452. usart\_receiver\_enable function**

Name	Description
Function name	usart_receiver_enable
Function prototype	void usart_receiver_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART receiver
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3...
Input parameter 2	new_state: Enable (TRUE) and disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable usart1 receiver */
usart_receiver_enable(USART1, TRUE);
```

## 5.20.7 usart\_clock\_config function

The table below describes the function usart\_clock\_config.

**Table 453. usart\_clock\_config function**

Name	Description
Function name	usart_clock_config
Function prototype	void usart_clock_config(usart_type* usart_x, usart_clock_polarity_type clk_pol, usart_clock_phase_type clk_ph, usart_lbc_type clk_lb);
Function description	Configure clock polarity and phase for synchronization feature
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3...
Input parameter 2	clk_pol: clock polarity for synchronization
Input parameter 3	clk_ph: clock phase for synchronization
Input parameter 4	clk_lb: selects whether to output clock on the last bit (upper bit) of data sent through synchronization feature
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### clk\_pol

Clock polarity selection.

USART\_CLOCK\_POLARITY\_LOW: Low

USART\_CLOCK\_POLARITY\_HIGH: High

### clk\_ph

Clock phase selection.

USART\_CLOCK\_PHASE\_1EDGE: 1<sup>st</sup> edge

USART\_CLOCK\_PHASE\_2EDGE: 2<sup>nd</sup> edge

### clk\_lb

Select whether to output clock on the last bit of data.

USART\_CLOCK\_LAST\_BIT\_NONE: No clock output

USART\_CLOCK\_LAST\_BIT\_OUTPUT: Clock output

### Example:

```
/* config synchronous mode */  
usart_clock_config(USART1, USART_CLOCK_POLARITY_HIGH, USART_CLOCK_PHASE_2EDGE,  
USART_CLOCK_LAST_BIT_OUTPUT);
```

## 5.20.8 usart\_clock\_enable function

The table below describes the function usart\_clock\_enable.

**Table 454. usart\_clock\_enable function**

Name	Description
Function name	usart_clock_enable
Function prototype	void usart_clock_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable clock output
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3...
Input parameter 2	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable clock */
usart_clock_enable(USART1, TRUE);
```

## 5.20.9 usart\_interrupt\_enable function

The table below describes the function usart\_interrupt\_enable.

**Table 455. usart\_interrupt\_enable function**

Name	Description
Function name	usart_interrupt_enable
Function prototype	void usart_interrupt_enable(usart_type* usart_x, uint32_t usart_int, confirm_state new_state);
Function description	Enable interrupts
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3...
Input parameter 2	usart_int: interrupt type
Input parameter 3	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**usart\_int**

Defines a peripheral interrupt.

- USART\_IDLE\_INT: Bus idle
- USART\_RDBF\_INT: Receive data buffer full
- USART\_TDC\_INT: Transmit data complete
- USART\_TDBE\_INT: Transmit data buffer empty
- USART\_PERR\_INT: Parity error
- USART\_BF\_INT: Break frame receive
- USART\_ERR\_INT: Error

USART\_CTSCF\_INT: CTS (Clear To Send) change

**Example:**

```
/* enable usart1 transmit complete interrupt */
usart_interrupt_enable (USART1, USART_TDC_INT, TRUE);
```

### 5.20.10 usart\_dma\_transmitter\_enable function

The table below describes the function usart\_dma\_transmitter\_enable.

**Table 456. usart\_dma\_transmitter\_enable function**

Name	Description
Function name	usart_dma_transmitter_enable
Function prototype	void usart_dma_transmitter_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable DMA transmitter
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3...
Input parameter 2	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable dma transmitter */
usart_dma_transmitter_enable (USART1, TRUE);
```

### 5.20.11 usart\_dma\_receiver\_enable function

The table below describes the function usart\_dma\_receiver\_enable.

**Table 457. usart\_dma\_receiver\_enable function**

Name	Description
Function name	usart_dma_receiver_enable
Function prototype	void usart_dma_receiver_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable DMA receiver
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3...
Input parameter 2	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable dma receiver */
usart_dma_receiver_enable (USART1, TRUE);
```

## 5.20.12 usart\_wakeup\_id\_set function

The table below describes the function usart\_wakeup\_id\_set.

**Table 458. usart\_wakeup\_id\_set function**

Name	Description
Function name	usart_wakeup_id_set
Function prototype	void usart_wakeup_id_set(usart_type* usart_x, uint8_t usart_id);
Function description	Set wakeup ID
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3...
Input parameter 2	usart_id: wakeup ID
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* config wakeup id */
usart_wakeup_id_set (USART1, 0x88);
```

## 5.20.13 usart\_wakeup\_mode\_set function

The table below describes the function usart\_wakeup\_mode\_set.

**Table 459. usart\_wakeup\_mode\_set function**

Name	Description
Function name	usart_wakeup_mode_set
Function prototype	void usart_wakeup_mode_set(usart_type* usart_x, usart_wakeup_mode_type wakeup_mode);
Function description	Set wakeup mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3
Input parameter 2	wakeup_mode: wakeup mode
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**wakeup\_mode**

Set wakeup mode to wake up from silent state.

USART\_WAKEUP\_BY\_IDLE\_FRAME: Woke up by idle frame

USART\_WAKEUP\_BY\_MATCHING\_ID: Woke up by ID matching

**Example:**

```
/* config usart1 wakeup mode */
usart_wakeup_mode_set (USART1, USART_WAKEUP_BY_MATCHING_ID);
```

## 5.20.14 usart\_receiver\_mute\_enable function

The table below describes the function usart\_receiver\_mute\_enable.

**Table 460. usart\_receiver\_mute\_enable function**

Name	Description
Function name	usart_receiver_mute_enable
Function prototype	void usart_receiver_mute_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable USART receiver mute mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3
Input parameter 2	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* config receiver mute */
usart_receiver_mute_enable (USART1, TRUE);
```

## 5.20.15 usart\_break\_bit\_num\_set function

The table below describes the function usart\_break\_bit\_num\_set.

**Table 461. usart\_break\_bit\_num\_set function**

Name	Description
Function name	usart_break_bit_num_set
Function prototype	void usart_break_bit_num_set(usart_type* usart_x, usart_break_bit_num_type break_bit);
Function description	Set USART break frame length
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3
Input parameter 2	break_bit: break frame length type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**break\_bit**

Set break frame length.

USART\_BREAK\_10BITS: 10 bits

USART\_BREAK\_11BITS: 11 bits

**Example:**

```
/* config break frame length 10bits */
usart_break_bit_num_set (USART1, USART_BREAK_10BITS);
```

## 5.20.16 usart\_lin\_mode\_enable function

The table below describes the function usart\_lin\_mode\_enable.

**Table 462. usart\_lin\_mode\_enable function**

Name	Description
Function name	usart_lin_mode_enable
Function prototype	void usart_lin_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable LIN mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3
Input parameter 2	new_state: Enable (TRUE) or disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable usart1 lin mode */  
usart_lin_mode_enable (USART1, TRUE);
```

## 5.20.17 usart\_data\_transmit function

The table below describes the function usart\_data\_transmit.

**Table 463. usart\_data\_transmit function**

Name	Description
Function name	usart_data_transmit
Function prototype	void usart_data_transmit(usart_type* usart_x, uint16_t data);
Function description	Transmit data
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3
Input parameter 2	Data: data to send
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* transmit data */  
uint16_t data = 0x88;  
usart_data_transmit (USART1, data);
```

## 5.20.18 usart\_data\_receive function

The table below describes the function usart\_data\_receive.

**Table 464. usart\_data\_receive function**

Name	Description
Function name	usart_data_receive
Function prototype	uint16_t usart_data_receive(usart_type* usart_x);
Function description	Receives data
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3
Input parameter 2	NA
Output parameter	NA
Return value	uint16_t: return the received data
Required preconditions	NA
Called functions	NA

**Example:**

```
/* receive data */  
uint16_t data = 0;  
data = usart_data_receive (USART1);
```

## 5.20.19 usart\_break\_send function

The table below describes the function usart\_break\_send.

**Table 465. usart\_break\_send function**

Name	Description
Function name	usart_break_send
Function prototype	void usart_break_send(usart_type* usart_x);
Function description	Sends break frame
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3
Input parameter 2	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* send break frame */  
usart_break_send (USART1);
```

## 5.20.20 usart\_smartcard\_guard\_time\_set function

The table below describes the function usart\_smartcard\_guard\_time\_set.

**Table 466. usart\_smartcard\_guard\_time\_set function**

Name	Description
Function name	usart_smartcard_guard_time_set
Function prototype	void usart_smartcard_guard_time_set(usart_type* usart_x, uint8_t guard_time_val);
Function description	Set smartcard guard time
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2, and USART3
Input parameter 2	guard_time_val: guard time, 0x00~0xFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* usart guard time set to 2 bit */
usart_smartcard_guard_time_set(USART1, 0x2);
```

## 5.20.21 usart\_irda\_smartcard\_division\_set function

The table below describes the function usart\_irda\_smartcard\_division\_set.

**Table 467. usart\_irda\_smartcard\_division\_set function**

Name	Description
Function name	usart_irda_smartcard_division_set
Function prototype	void usart_irda_smartcard_division_set(usart_type* usart_x, uint8_t div_val);
Function description	Infrared and smartcard frequency division settings
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	div_val: division value
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* usart clock set to (apbclk / (2 * 20)) */
usart_irda_smartcard_division_set(USART1, 20);
```

## 5.20.22 usart\_smartcard\_mode\_enable function

The table below describes the function usart\_smartcard\_mode\_enable.

**Table 468. usart\_smartcard\_mode\_enable function**

Name	Description
Function name	usart_smartcard_mode_enable
Function prototype	void usart_smartcard_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable smartcode mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: Enable (TRUE), Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable the smartcard mode */
usart_smartcard_mode_enable(USART1, TRUE);
```

## 5.20.23 usart\_smartcard\_nack\_set function

The table below describes the function usart\_smartcard\_nack\_set.

**Table 469. usart\_smartcard\_nack\_set function**

Name	Description
Function name	usart_smartcard_nack_set
Function prototype	void usart_smartcard_nack_set(usart_type* usart_x, confirm_state new_state);
Function description	Enable smartcard NACK
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: Enable (TRUE), Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable the nack transmission */
usart_smartcard_nack_set(USART1, TRUE);
```

## 5.20.24 usart\_single\_line\_halfduplex\_select function

The table below describes the function usart\_single\_line\_halfduplex\_select.

**Table 470. usart\_single\_line\_halfduplex\_select function**

Name	Description
Function name	usart_single_line_halfduplex_select
Function prototype	void usart_single_line_halfduplex_select(usart_type* usart_x, confirm_state new_state);
Function description	Enable single-wire half-duplex mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: Enable (TRUE), Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable halfduplex */
usart_single_line_halfduplex_select(USART1, TRUE);
```

## 5.20.25 usart\_irda\_mode\_enable function

The table below describes the function usart\_irda\_mode\_enable.

**Table 471. usart\_irda\_mode\_enable function**

Name	Description
Function name	usart_irda_mode_enable
Function prototype	void usart_irda_mode_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable infrared mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: Enable (TRUE), Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable irda mode */
usart_irda_mode_enable(USART1, TRUE);
```

## 5.20.26 usart\_irda\_low\_power\_enable function

The table below describes the function usart\_irda\_low\_power\_enable.

**Table 472. usart\_irda\_low\_power\_enable function**

Name	Description
Function name	usart_irda_low_power_enable
Function prototype	void usart_irda_low_power_enable(usart_type* usart_x, confirm_state new_state);
Function description	Enable infrared low-power mode
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	new_state: Enable (TRUE), Disable (FALSE)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* enable irda lowpower mode */
usart_irda_low_power_enable (USART1, TRUE);
```

## 5.20.27 usart\_hardware\_flow\_control\_set function

The table below describes the function usart\_hardware\_flow\_control\_set.

**Table 473. usart\_hardware\_flow\_control\_set function**

Name	Description
Function name	usart_hardware_flow_control_set
Function prototype	void usart_hardware_flow_control_set(usart_type* usart_x, usart_hardware_flow_control_type flow_state);
Function description	Set peripheral flow control
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	flow_state: flow control type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**flow\_state**

USART\_HARDWARE\_FLOW\_NONE: No hardware flow control

USART\_HARDWARE\_FLOW\_RTS: RTS

USART\_HARDWARE\_FLOW\_CTS: CTS

USART\_HARDWARE\_FLOW\_RTS\_CTS: RTS and CTS

**Example:**

```
/* hardware flow set none */
usart_hardware_flow_control_set (USART1, USART_HARDWARE_FLOW_NONE);
```

## 5.20.28 usart\_flag\_get function

The table below describes the function usart\_flag\_get.

Table 474. usart\_flag\_get function

Name	Description
Function name	usart_flag_get
Function prototype	flag_status usart_flag_get(usart_type* usart_x, uint32_t flag);
Function description	Get flag status
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	Flag: flag selection
Output parameter	NA
Return value	flag_status: SET or RESET
Required preconditions	NA
Called functions	NA

### flag

USART_CTSCF_FLAG:	CTS (Clear To Send) change flag
USART_BFF_FLAG:	Break frame receive flag
USART_TDBE_FLAG:	Transmit buffer empty flag
USART_TDC_FLAG:	Transmit complete flag
USART_RDBF_FLAG:	Receive data buffer full flag
USART_IDLEF_FLAG:	Idle frame flag
USART_ROERR_FLAG:	Receive overflow flag
USART_NERR_FLAG:	Noise error flag
USART_FERR_FLAG:	Frame error flag
USART_PERR_FLAG:	Parity error flag

### Example:

```
/* wait data transmit complete flag */
while(usart_flag_get (USART1, USART_TDC_FLAG) == RESET);
```

## 5.20.29 usart\_interrupt\_flag\_get function

The table below describes the function usart\_interrupt\_flag\_get.

**Table 475. usart\_interrupt\_flag\_get function**

Name	Description
Function name	usart_interrupt_flag_get
Function prototype	flag_status usart_interrupt_flag_get(usart_type* usart_x, uint32_t flag);
Function description	Check whether the selected flag is set or not
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2...
Input parameter 2	flag: flag selection
Output parameter	NA
Return value	flag_status: SET or RESET
Required preconditions	NA
Called functions	NA

### flag

USART_CTSCF_FLAG:	CTS (Clear To Send) change flag
USART_BFF_FLAG:	Break frame receive flag
USART_TDBE_FLAG:	Transmit buff empty flag
USART_TDC_FLAG:	Transmit complete flag
USART_RDBF_FLAG:	Receive data buffer full flag
USART_IDLEF_FLAG:	Idle frame flag
USART_ROERR_FLAG:	Receive overflow flag
USART_NERR_FLAG:	Noise error flag
USART_FERR_FLAG:	Frame error flag
USART_PERR_FLAG:	Parity error flag

### Example

```
/* check received data flag */
if(usart_interrupt_flag_get(USART1, USART_RDBF_FLAG) != RESET)
{
}
```

### 5.20.30 usart\_flag\_clear function

The table below describes the function usart\_flag\_clear.

Table 476. usart\_flag\_clear function

Name	Description
Function name	usart_flag_clear
Function prototype	void usart_flag_clear(usart_type* usart_x, uint32_t flag);
Function description	Clear flag
Input parameter 1	usart_x: indicates the selected peripheral, it can be USART1, USART2 and USART3
Input parameter 2	Flag: clear the selected flag
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### flag

- USART\_CTSCF\_FLAG: CTS (Clear To Send) change flag  
USART\_BFF\_FLAG: Break frame receive flag  
USART\_TDC\_FLAG: Transmit complete flag  
USART\_RDBF\_FLAG: Receive data buffer full flag

#### Example:

```
/* clear data transmit complete flag */  
usart_flag_clear (USART1, USART_TDC_FLAG );
```

## 5.21 Watchdog timer (WDT)

The WDT register structure `wdt_type` is defined in the “`at32f403_wdt.h`”:

```
/**  
 * @brief type define wdt register all  
 */  
  
typedef struct  
{  
  
} wdt_type;
```

The table below gives a list of the WDT registers.

**Table 477. Summary of WDT registers**

Register	Description
cmd	Command register
div	Divider register
rld	Reload register
sts	Status register

The table below gives a list of WDT library functions.

**Table 478. Summary of WDT library functions**

Function name	Description
wdt_enable	Enable watchdog
wdt_counter_reload	Reload counter
wdt_reload_value_set	Set reload value
wdt_divider_set	Set division value
wdt_register_write_enable	Unlock WDT_DIV and WDT_RLD register write protection
wdt_flag_get	Get flag

## 5.21.1 wdt\_enable function

The table below describes the function wdt\_enable.

**Table 479. wdt\_enable function**

Name	Description
Function name	wdt_enable
Function prototype	void wdt_enable(void);
Function description	Enable watchdog
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
wdt_enable();
```

## 5.21.2 wdt\_counter\_reload function

The table below describes the function wdt\_counter\_reload.

**Table 480. wdt\_counter\_reload function**

Name	Description
Function name	wdt_counter_reload
Function prototype	void wdt_counter_reload(void);
Function description	Reload counter
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
wdt_counter_reload();
```

### 5.21.3 `wdt_reload_value_set` function

The table below describes the function `wdt_reload_value_set`.

**Table 481. `wdt_reload_value_set` function**

Name	Description
Function name	<code>wdt_reload_value_set</code>
Function prototype	<code>void wdt_reload_value_set(uint16_t reload_value);</code>
Function description	Set reload value
Input parameter	reload_value: reload value, 0x000~0xFFFF
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
wdt_reload_value_set(0xFFFF);
```

### 5.21.4 `wdt_divider_set` function

The table below describes the function `wdt_divider_set`.

**Table 482. `wdt_divider_set` function**

Name	Description
Function name	<code>wdt_divider_set</code>
Function prototype	<code>void wdt_divider_set(wdt_division_type division);</code>
Function description	Set division value
Input parameter	Division: watchdog division value Refer to the “division” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### division

Select watchdog division value.

`WDT_CLK_DIV_4`: Divided by 4

`WDT_CLK_DIV_8`: Divided by 8

`WDT_CLK_DIV_16`: Divided by 16

`WDT_CLK_DIV_32`: Divided by 32

`WDT_CLK_DIV_64`: Divided by 64

`WDT_CLK_DIV_128`: Divided by 128

`WDT_CLK_DIV_256`: Divided by 256

**Example:**

```
wdt_divider_set(WDT_CLK_DIV_4);
```

## 5.21.5 wdt\_register\_write\_enable function

The table below describes the function wdt\_register\_write\_enable.

**Table 483. wdt\_register\_write\_enable function**

Name	Description
Function name	wdt_register_write_enable
Function prototype	void wdt_register_write_enable( confirm_state new_state);
Function description	Unlock WDT_DIV and WDT_RLD write protection
Input parameter	new_state: unlock register write protection This parameter can be TRUE or FALSE.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
wdt_register_write_enable(TRUE);
```

## 5.21.6 wdt\_flag\_get function

The table below describes the function wdt\_flag\_get.

**Table 484. wdt\_flag\_get function**

Name	Description
Function name	wdt_flag_get
Function prototype	flag_status wdt_flag_get(uint16_t wdt_flag);
Function description	Get flag
Input parameter	Flag: flag selection Refer to the "flag" description below for details.
Output parameter	NA
Return value	flag_status: flag status This parameter can be SET or RESET.
Required preconditions	NA
Called functions	NA

### flag

This is used for flag selection, including:

WDT\_DIVF\_UPDATE\_FLAG: Division value update complete

WDT\_RLDF\_UPDATE\_FLAG: Reload value update complete

**Example:**

```
wdt_flag_get(WDT_DIVF_UPDATE_FLAG);
```

## 5.22 Window watchdog timer (WWDT)

The WWDT register structure wwdt\_type is defined in the “at32f403\_wwdt.h”:

```
/*
 * @brief type define wwdt register all
 */
typedef struct
{

} wwdt_type;
```

The table below gives a list of the WWDT registers.

**Table 485. Summary of WWDT registers**

Register	Description
ctrl	Control register
cfg	Configuration register
sts	Status register

The table below gives a list of WWDT library functions.

**Table 486. Summary of WWDT library functions**

Function name	Description
wwdt_reset	Reset window watchdog registers
wwdt_divider_set	Set divider
wwdt_flag_clear	Clear reload counter interrupt flag
wwdt_enable	Enable WWDT
wwdt_interrupt_enable	Enable reload counter interrupt
wwdt_flag_get	Get flag
wwdt_counter_set	Set counter value
wwdt_window_counter_set	Set window value

## 5.22.1 wwdt\_reset function

The table below describes the function wwdt\_reset.

**Table 487. wwdt\_reset function**

Name	Description
Function name	wwdt_reset
Function prototype	void wwdt_reset(void);
Function description	Reset window watchdog registers to their initial values.
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	void crm_periph_reset(crm_periph_reset_type value, confirm_state new_state);

**Example:**

```
wwdt_reset();
```

## 5.22.2 wwdt\_divider\_set function

The table below describes the function wwdt\_divider\_set.

**Table 488. wwdt\_divider\_set function**

Name	Description
Function name	wwdt_divider_set
Function prototype	void wwdt_divider_set(wwdt_division_type division);
Function description	Set divider
Input parameter	Division: WWDT division value Refer to the “division” description below for details.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### division

Select WWDT division value.

WWDT\_PCLK1\_DIV\_4096: Divided by 4096

WWDT\_PCLK1\_DIV\_8192: Divided by 8192

WWDT\_PCLK1\_DIV\_16384: Divided by 16384

WWDT\_PCLK1\_DIV\_32768: Divided by 32768

**Example:**

```
wwdt_divider_set(WWDT_PCLK1_DIV_4096);
```

### 5.22.3 wwdt\_enable function

The table below describes the function wwdt\_enable.

Table 489. wwdt\_enable function

Name	Description
Function name	wwdt_enable
Function prototype	void wwdt_enable(uint8_t wwdt_cnt);
Function description	Enable WWDT
Input parameter	wwdt_cnt: WWDT counter initial value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
wwdt_enable(0x7F);
```

### 5.22.4 wwdt\_interrupt\_enable function

The table below describes the function wwdt\_interrupt\_enable.

Table 490. wwdt\_interrupt\_enable function

Name	Description
Function name	wwdt_interrupt_enable
Function prototype	void wwdt_interrupt_enable(void);
Function description	Enable reload counter interrupt
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
wwdt_interrupt_enable();
```

### 5.22.5 wwdt\_counter\_set function

The table below describes the function wwdt\_counter\_set.

Table 491. wwdt\_counter\_set function

Name	Description
Function name	wwdt_counter_set
Function prototype	void wwdt_counter_set(uint8_t wwdt_cnt);
Function description	Set counter value
Input parameter	wwdt_cnt: WWDT counter value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
wwdt_counter_set(0x7F);
```

## 5.22.6 wwdt\_window\_counter\_set function

The table below describes the function wwdt\_window\_counter\_set.

**Table 492. wwdt\_window\_counter\_set function**

Name	Description
Function name	wwdt_window_counter_set
Function prototype	void wwdt_window_counter_set(uint8_t window_cnt);
Function description	Set window value
Input parameter	wwdt_cnt: WWDT window value, 0x40~0x7F
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
wwdt_window_counter_set(0x6F);
```

## 5.22.7 wwdt\_flag\_get function

The table below describes the function wwdt\_flag\_get.

**Table 493. wwdt\_flag\_get function**

Name	Description
Function name	wwdt_flag_get
Function prototype	flag_status wwdt_flag_get(void);
Function description	Get reload counter interrupt flag
Input parameter	NA
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET
Required preconditions	NA
Called functions	NA

**Example:**

```
wwdt_flag_get();
```

## 5.22.8 wwdt\_interrupt\_flag\_get function

The table below describes the function wwdt\_interrupt\_flag\_clear.

**Table 494. wwdt\_interrupt\_flag\_clear function**

Name	Description
Function name	wwdt_interrupt_flag_get
Function prototype	flag_status wwdt_interrupt_flag_get(void);
Function description	Get reload counter interrupt flag and judge the corresponding interrupt enable bit
Input parameter 1	NA
Output parameter	NA
Return value	flag_status: flag status Return SET or RESET.
Required preconditions	NA
Called functions	NA

**Example**

```
wwdt_interrupt_flag_get();
```

## 5.22.9 wwdt\_flag\_clear function

The table below describes the function wwdt\_flag\_clear.

**Table 495. wwdt\_flag\_clear function**

Name	Description
Function name	wwdt_flag_clear
Function prototype	void wwdt_flag_clear(void);
Function description	Clear reload counter interrupt flag
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
wwdt_flag_clear();
```

## 5.23 External memory controller (XMC)

For XMC bank1 chip-select control register and timing register, their register structure

xmc\_bank1\_ctrl\_tmg\_reg\_type is defined in the “at32f403\_xmc.h”:

```
/**  
 * @brief type define xmc bank1 ctrl and tmg register  
 */  
typedef struct  
{  
    ...  
  
} xmc_bank1_ctrl_tmg_reg_type;
```

XMC bank1 write timing register structure xmc\_bank1\_tmgwr\_reg\_type is defined in the “at32f403\_xmc.h”:

```
/**  
 * @brief type define xmc bank1 tmgwr register  
 */  
typedef struct  
{  
    ...  
  
} xmc_bank1_tmgwr_reg_type;
```

XMC bank1 register structure xmc\_bank1\_type is defined in the “at32f403\_xmc.h”:

```
/**  
 * @brief xmc bank1 registers  
 */  
typedef struct  
{  
    ...  
  
} xmc_bank1_type;
```

XMC bank2 register structure xmc\_bank2\_type is defined in the “at32f403\_xmc.h”:

```
/**  
 * @brief xmc bank2 registers  
 */  
typedef struct  
{  
    ...  
  
} xmc_bank2_type;
```

XMC bank3 register structure xmc\_bank3\_type is defined in the “at32f403\_xmc.h”.

```
/**  
 * @brief xmc bank3 registers  
 */
```

```

typedef struct
{
    ...
} xmc_bank3_type;

```

XMC bank4 register structure xmc\_bank4\_type is defined in the “at32f403\_xmc.h”.

```

/** @brief xmc bank4 registers
 */
typedef struct
{
    ...
} xmc_bank4_type;

```

The table below gives a list of the XMC registers.

**Table 496. Summary of XMC registers**

Register	Description
xmc_bk1ctrl1	SRAM/NOR Flash chip select control register 1
xmc_bk1tmg1	SRAM/NOR Flash chip select timing register 1
xmc_bk1ctrl2	SRAM/NOR Flash chip select control register 2
xmc_bk1tmg2	SRAM/NOR Flash chip select timing register 2
xmc_bk1ctrl3	SRAM/NOR Flash chip select control register 3
xmc_bk1tmg3	SRAM/NOR Flash chip select timing register 3
xmc_bk1ctrl4	SRAM/NOR Flash chip select control register 4
xmc_bk1tmg4	SRAM/NOR Flash chip select timing register 4
xmc_bk2ctrl	Nand Flash control register 2
xmc_bk2is	Interrupt enable and FIFO status register 2
xmc_bk2tmgrg	Regular memory timing register 2
xmc_bk2tmgsp	Special memory timing register 2
xmc_bk2ecc	ECC value register2
xmc_bk3ctrl	Nand Flash control register 3
xmc_bk3is	Interrupt enable and FIFO status register 3
xmc_bk3tmgrg	Common space timing register 3
xmc_bk3tmgsp	Special space timing register 3
xmc_bk3ecc	ECC result register 3
xmc_bk4ctrl	PC card control register
xmc_bk4is	Interrupt enable and FIFO status register 4
xmc_bk4tmgcm	Common space timing register 4
xmc_bk4tmgat	Attribute space timing register 4
xmc_bk4tmgio	IO space timing register 4
xmc_bk1tmgwr1	SRAM/NOR Flash write timing register 1
xmc_bk1tmgwr2	SRAM/NOR Flash write timing register 2

Register	Description
xmc_bk1tmgwr3	SRAM/NOR Flash write timing register 3
xmc_bk1tmgwr4	SRAM/NOR Flash write timing register 4
xmc_ext1	SRAM/NOR Flash extra timing register 1
xmc_ext2	SRAM/NOR extra extend register 2
xmc_ext3	SRAM/NOR extra extend register 3
xmc_ext4	SRAM/NOR Flash extra timing register 4

The table below gives a list of XMC library functions.

**Table 497. Summary of XMC library functions**

Function name	Description
xmc_nor_sram_reset	Reset nor/sram controller
xmc_nor_sram_init	Initialize nor/sram controller
xmc_nor_sram_timing_config	Configure nor/sram controller timing
xmc_norsram_default_para_init	Initialize the parameters of the xmc_nor_sram_init_struct
xmc_norsram_timing_default_para_init	Initialize the parameters of the xmc_rw_timing_struct and xmc_w_timing_struct
xmc_nor_sram_enable	Enable nor/sram controller
xmc_ext_timing_config	Configure nor/sram extension controller
xmc_nand_reset	Reset nand controller
xmc_nand_init	Initialize nand controller
xmc_nand_timing_config	Configure nand controller timing
xmc_nand_default_para_init	Initialize the parameters of the xmc_nand_init_struct
xmc_nand_timing_default_para_init	Initialize the parameters of the xmc_common_spacetiming_struct and xmc_attribute_spacetiming_struct
xmc_nand_enable	Enable nand controller
xmc_nand_ecc_enable	Enable nand controller ECC function
xmc_ecc_get	Get ECC value
xmc_interrupt_enable	Enable XMC interrupts
xmc_flag_status_get	Get XMC interrupt flags
xmc_flag_clear	Clear XMC interrupt flags
xmc_pccard_reset	Reset pccard controller
xmc_pccard_init	Initialize pccard controller
xmc_pccard_timing_config	Configure pccard controller timing
xmc_pccard_default_para_init	Initialize the parameters of the xmc_pccard_init_struct
xmc_pccard_timing_default_para_init	Initialize the parameters of the xmc_common_spacetiming_struct, xmc_attribute_spacetiming_struct and xmc_iospace_timing_struct
xmc_pccard_enable	Enable pccard controller

### 5.23.1 xmc\_nor\_sram\_reset function

The table below describes the function xmc\_nor\_sram\_reset.

**Table 498. xmc\_nor\_sram\_reset function**

Name	Description
Function name	xmc_nor_sram_reset
Function prototype	void xmc_nor_sram_reset(xmc_nor_sram_subbank_type xmc_subbank);
Function description	Reset nor/sram controller
Input parameter	xmc_subbank: subbank selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### xmc\_subbank

Select a subbank to reset.

XMC\_BANK1\_NOR\_SRAM1: xmc\_subbank1

XMC\_BANK1\_NOR\_SRAM2: xmc\_subbank2

XMC\_BANK1\_NOR\_SRAM3: xmc\_subbank3

XMC\_BANK1\_NOR\_SRAM4: xmc\_subbank4

#### Example:

```
/* reset nor/sram subbank1 */
xmc_nor_sram_reset(XMC_BANK1_NOR_SRAM1);
```

### 5.23.2 xmc\_nor\_sram\_init function

The table below describes the function xmc\_nor\_sram\_init.

**Table 499. xmc\_nor\_sram\_init function**

Name	Description
Function name	xmc_nor_sram_init
Function prototype	void xmc_nor_sram_init(xmc_norsram_init_type* xmc_norsram_init_struct);
Function description	Initialize nor/sram controller
Input parameter 1	xmc_norsram_init_struct: xmc_norsram_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### xmc\_norsram\_init\_type struct

xmc\_norsram\_init\_type is defined in the at32f403\_xmc.h:

typedef struct

```
{
    xmc_nor_sram_subbank_type    subbank;
    xmc_data_addr_mux_type      data_addr_multiplex;
    xmc_memory_type              device;
    xmc_data_width_type          bus_type;
    xmc_burst_access_mode_type   burst_mode_enable;
```

```
xmc_asyn_wait_type      asynwait_enable;
xmc_wait_signal_polarity_type wait_signal_lv;
xmc_wrap_mode_type       wrapped_mode_enable;
xmc_wait_timing_type     wait_signal_config;
xmc_write_operation_type write_enable;
xmc_wait_signal_type     wait_signal_enable;
xmc_extended_mode_type  write_timing_enable;
xmc_write_burst_type    write_burst_syn;
} xmc_norsram_init_type;
```

**subbank**

Select a subbank to reset.

XMC\_BANK1\_NOR\_SRAM1: xmc subbank1  
XMC\_BANK1\_NOR\_SRAM2: xmc subbank2  
XMC\_BANK1\_NOR\_SRAM3: xmc subbank3  
XMC\_BANK1\_NOR\_SRAM4: xmc subbank4

**data\_addr\_multiplex**

Define whether the lower 16 bits of the xmc address line is multiplexed with data line or not.

XMC\_DATA\_ADDR\_MUX\_DISABLE: Not multiplexed  
XMC\_DATA\_ADDR\_MUX\_ENABLE: Multiplexed

**Device**

Select an external memory device.

XMC\_DEVICE\_SRAM: sram  
XMC\_DEVICE\_PSRAM: psram  
XMC\_DEVICE\_NOR: nor

**bus\_type**

xmc data bus bit width.

XMC\_BUSTYPE\_8\_BITS: 8-bit data bus  
XMC\_BUSTYPE\_16\_BITS: 1-6bit data bus

**burst\_mode\_enable**

Enable or disable burst mode.

XMC\_BURST\_MODE\_DISABLE: Burst mode disabled  
XMC\_BURST\_MODE\_ENABLE: Burst mode enabled

**asynwait\_enable**

Enable or disable wait signals during asynchronous transmission.

XMC\_ASYN\_WAIT\_DISABLE: Disabled  
XMC\_ASYN\_WAIT\_ENABLE: Enabled

**wait\_signal\_lv**

Select the polarity of wait signals. This bit is used to set the polarity of NWAIT signal in synchronous mode.

XMC\_WAIT\_SIGNAL\_LEVEL\_LOW: Active low  
XMC\_WAIT\_SIGNAL\_LEVEL\_HIGH: Active high

**wrapped\_mode\_enable**

In synchronous mode, this bit is used to define whether the XMC controller will or not split an not-aligned AHB access into two accesses.

XMC\_WWRAPPED\_MODE\_DISABLE: Direct wrapped burst mode is disabled

XMC\_WRAPPED\_MODE\_ENABLE: Direct wrapped burst mode is enabled

#### **wait\_signal\_config**

Wait timing configuration. Valid only in synchronous mode.

XMC\_WAIT\_SIGNAL\_SYN\_BEFORE: NWAIT signal is active one data cycle before wait state

XMC\_WAIT\_SIGNAL\_SYN\_DURING: NWAIT signal is active during wait state

#### **write\_enable**

Write enable bit.

XMC\_WRITE\_OPERATION\_DISABLE: Write operation is disabled

XMC\_WRITE\_OPERATION\_ENABLE: Write operation is enabled

#### **wait\_signal\_enable**

Wait signal enable bit during synchronous transmission.

XMC\_WAIT\_SIGNAL\_DISABLE: NWAIT signal disabled

XMC\_WAIT\_SIGNAL\_ENABLE: NWAIT signal enabled

#### **write\_timing\_enable**

Wait timing enable bit. Read memory and write memory operate at different timings. In other words, SRAM/NOR write timing register (XMC\_BKxTMGWR) is released.

XMC\_WRITE\_TIMING\_DISABLE: Read and write timings are the same

XMC\_WRITE\_TIMING\_ENABLE: Read and write timings are different

#### **write\_burst\_syn**

Write burst enable bit

XMC\_WRITE\_BURST\_SYN\_DISABLE: Write operation is performed in asynchronous mode

XMC\_WRITE\_BURST\_SYN\_ENABLE: Write operation is performed in synchronous mode

#### **Example:**

```
xmc_norsram_init_type xmc_norsram_init_struct;  
xmc_norsram_init_struct.subbank = XMC_BANK1_NOR_SRAM1;  
xmc_norsram_init_struct.data_addr_mux_enable = XMC_DATA_ADDR_MUX_ENABLE;  
xmc_norsram_init_struct.device = XMC_DEVICE_PSRAM;  
xmc_norsram_init_struct.bus_type = XMC_BUSTYPE_16_BITS;  
xmc_norsram_init_struct.burst_mode_enable = XMC_BURST_MODE_DISABLE;  
xmc_norsram_init_struct.asynwait_enable = XMC_ASYN_WAIT_DISABLE;  
xmc_norsram_init_struct.wait_signal_lv = XMC_WAIT_SIGNAL_LEVEL_LOW;  
xmc_norsram_init_struct.wrapped_mode_enable = XMC_WRAPPED_MODE_DISABLE;  
xmc_norsram_init_struct.wait_signal_config = XMC_WAIT_SIGNAL_SYN_BEFORE;  
xmc_norsram_init_struct.write_enable = XMC_WRITE_OPERATION_ENABLE;  
xmc_norsram_init_struct.wait_signal_enable = XMC_WAIT_SIGNAL_DISABLE;  
xmc_norsram_init_struct.write_burst_syn = XMC_WRITE_BURST_SYN_DISABLE;  
xmc_norsram_init_struct.write_timing_enable = XMC_WRITE_TIMING_DISABLE;  
xmc_nor_sram_init(&xmc_norsram_init_struct);
```

### 5.23.3 xmc\_nor\_sram\_timing\_config function

Table 51describes the function xmc\_nor\_sram\_timing\_config.

**Table 500. scfg\_mem\_map\_get function**

Name	Description
Function name	xmc_nor_sram_timing_config
Function prototype	void xmc_nor_sram_timing_config(xmc_norsram_timing_init_type* xmc_rw_timing_struct, xmc_norsram_timing_init_type* xmc_w_timing_struct);
Function description	Configure the nor/sram controller timing
Input parameter 1	xmc_rw_timing_struct: xmc_norsram_timing_init_type pointer It is used to configure read/write timing when the write timing is not enabled independently.
Input parameter 2	xmc_w_timing_struct: xmc_norsram_timing_init_type pointer It is used to configure write timing when the write timing is enabled independently.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### xmc\_norsram\_timing\_init\_type struct

xmc\_norsram\_timing\_init\_type is defined in the at32f403\_xmc.h:

typedef struct

```
{
    xmc_nor_sram_subbank_type    subbank;
    xmc_extended_mode_type      write_timing_enable;
    uint32_t                      addr_setup_time;
    uint32_t                      addr_hold_time;
    uint32_t                      data_setup_time;
    uint32_t                      bus_latency_time;
    uint32_t                      clk_psc;
    uint32_t                      data_latency_time;
    xmc_access_mode_type         mode;
} xmc_norsram_timing_init_type;
```

#### subbank

Select a subbank to reset.

XMC\_BANK1\_NOR\_SRAM1: xmc subbank1

XMC\_BANK1\_NOR\_SRAM2: xmc subbank2

XMC\_BANK1\_NOR\_SRAM3: xmc subbank3

XMC\_BANK1\_NOR\_SRAM4: xmc subbank4

#### write\_timing\_enable

Wait timing enable bit. Read memory and write memory operate at different timings. In other words, SRAM/NOR write timing register (XMC\_BKxTMGWR) is released.

XMC\_WRITE\_TIMING\_DISABLE: Read and write timings are the same

XMC\_WRITE\_TIMING\_ENABLE: Read and write timings are different

#### addr\_setup\_time

Address setup time.

0000: 0 HCLK clock cycle added

0001: 1 HCLK clock cycle added

.....

1111: 15 HCLK clock cycles added

#### **addr\_hold\_time**

Address hold time.

0000: 0 HCLK clock cycle added

0001: 1 HCLK clock cycles added

.....

1111: 15 HCLK clock cycles added

#### **data\_setup\_time**

Data setup time

0000: 0 HCLK clock cycle added

0001: 1 HCLK clock cycle added

.....

1111: 15 HCLK clock cycles added

#### **bus\_latency\_time**

Bus latency time. In order to avoid conflicts on data bus, in multiplexed mode or synchronous mode, the XMC controller inserts latency time into data bus after a single read operation.

0000: 1 HCLK clock cycle added

0001: 2 HCLK clock cycles added

.....

1111: 16 HCLK clock cycles added

#### **clk\_psc**

Clock frequency division factor. Valid only in synchronous mode. It is used to define the clock frequency of the XMC\_CLK.

0000: Reserved

0001: XMC\_CLK cycles= 2 x HCLK clock cycles

0010: XMC\_CLK cycles= 3 x HCLK clock cycles

.....

1111: XMC\_CLK cycles= 16 x HCLK clock cycles

#### **data\_latency\_time**

Data latency time. Valid only in synchronous mode.

0000: 0 HCLK clock cycle added

0001: 1 HCLK clock cycle added

.....

1111: 15 HCLK clock cycles added

#### **Mode**

Asynchronous access mode selection bit. Valid only when the RWTD bit is enabled.

00: Mode A

01: Mode B

10: Mode C

11: Mode D

#### **Example:**

```
xmc_norsram_timing_init_type rw_timing_struct, w_timing_struct;
```

```

rw_timing_struct.subbank          = XMC_BANK1_NOR_SRAM1;
rw_timing_struct.mode             = XMC_ACCESS_MODE_A;
rw_timing_struct.write_timing_enable = XMC_WRITE_TIMING_DISABLE;
rw_timing_struct.addr_hold_time   = 0x08;
rw_timing_struct.addr_setup_time  = 0x09;
rw_timing_struct.data_setup_time  = 0x0F;
rw_timing_struct.data_latency_time = 0x0;
rw_timing_struct.bus_latency_time = 0x0;
rw_timing_struct.clk_psc          = 0x0;

w_timing_struct.subbank          = XMC_BANK1_NOR_SRAM1;
w_timing_struct.mode             = XMC_ACCESS_MODE_A;
w_timing_struct.write_timing_enable = XMC_WRITE_TIMING_DISABLE;
w_timing_struct.addr_hold_time   = 0x08;
w_timing_struct.addr_setup_time  = 0x09;
w_timing_struct.data_setup_time  = 0x0F;
w_timing_struct.data_latency_time = 0x0;
w_timing_struct.bus_latency_time = 0x0;
w_timing_struct.clk_psc          = 0x0;

xmc_nor_sram_timing_config(&rw_timing_struct, &w_timing_struct);

```

#### 5.23.4 xmc\_norsram\_default\_para\_init function

The table below describes the function xmc\_norsram\_default\_para\_init.

**Table 501. xmc\_norsram\_default\_para\_init function**

Name	Description
Function name	xmc_norsram_default_para_init
Function prototype	void xmc_norsram_default_para_init(xmc_norsram_init_type* xmc_nor_sram_init_struct);
Function description	Initialize the parameters of the xmc_nor_sram_init_struct
Input parameter	xmc_nor_sram_init_struct: xmc_norsram_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Table 521 shows the values of members in the xmc\_nor\_sram\_init\_struct:

**Table 502. xmc\_nor\_sram\_init\_struct default values**

Member	Default values
subbank	XMC_BANK1_NOR_SRAM1
data_addr_mux	XMC_DATA_ADDR_MUX_ENABLE
device	XMC_DEVICE_SRAM
bus_type	XMC_BUSTYPE_8_BITS
burst_mode_enable	XMC_BURST_MODE_DISABLE
asynwait_enable	XMC_ASYN_WAIT_DISABLE
wait_signal_lv	XMC_WAIT_SIGNAL_LEVEL_LOW
wrapped_mode_enable	XMC_WWRAPPED_MODE_DISABLE
wait_signal_config	XMC_WAIT_SIGNAL_SYN_BEFORE
write_enable	XMC_WRITE_OPERATION_ENABLE
wait_signal_enable	XMC_WAIT_SIGNAL_ENABLE
write_timing_enable	XMC_WRITE_TIMING_DISABLE
write_burst_syn	XMC_WRITE_BURST_SYN_DISABLE

**Example:**

```
xmc_norsram_init_type xmc_norsram_init_struct;
/* fill each xmc_nor_sram_init_struct member with its default value */
xmc_norsram_default_para_init(&xmc_norsram_init_struct);
```

### 5.23.5 xmc\_norsram\_timing\_default\_para\_init function

The table below describes the function xmc\_norsram\_timing\_default\_para\_init.

**Table 503. xmc\_norsram\_timing\_default\_para\_init function**

Name	Description
Function name	xmc_norsram_timing_default_para_init
Function prototype	void xmc_norsram_timing_default_para_init(xmc_norsram_timing_init_type* xmc_rw_timing_struct, xmc_norsram_timing_init_type* xmc_w_timing_struct);
Function description	Initialize the parameters of the xmc_rw_timing_struct and xmc_w_timing_struct
Input parameter	xmc_rw_timing_struct: xmc_norsram_timing_init_type pointer xmc_w_timing_struct: xmc_norsram_timing_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below shows the values of members in the xmc\_rw\_timing\_struct and xmc\_w\_timing\_struct:

**Table 504. xmc\_rw\_timing\_struct and xmc\_w\_timing\_struct**

Member	Default values
subbank	XMC_BANK1_NOR_SRAM1
write_timing_enable	XMC_WRITE_TIMING_DISABLE
addr_setup_time	0xF
addr_hold_time	0xF
data_setup_time	0xFF
bus_latency_time	0xF
clk_psc	0xF
data_latency_time	0xF
mode	XMC_ACCESS_MODE_A

**Example:**

```
xmc_norsram_timing_init_type rw_timing_struct, w_timing_struct;
/* fill each xmc_rw_timing_struct and xmc_w_timing_struct member with its default value */
xmc_norsram_timing_default_para_init (&xmc_rw_timing_struct, &xmc_w_timing_struct);
```

## 5.23.6 xmc\_nor\_sram\_enable function

The table below describes the function xmc\_nor\_sram\_enable.

**Table 505. xmc\_nor\_sram\_enable function**

Name	Description
Function name	xmc_nor_sram_enable
Function prototype	void xmc_nor_sram_enable(xmc_nor_sram_subbank_type xmc_subbank, confirm_state new_state);
Function description	Enable nor/sram controller
Input parameter 1	xmc_subbank: subbank selection
Input parameter 2	new_state: enable or disable
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**xmc\_subbank**

Select a subbank.

XMC\_BANK1\_NOR\_SRAM1: xmc subbank1

XMC\_BANK1\_NOR\_SRAM2: xmc subbank2

XMC\_BANK1\_NOR\_SRAM3: xmc subbank3

XMC\_BANK1\_NOR\_SRAM4: xmc subbank4

**new\_state**

FALSE: Disable controller

TRUE: Enable controller

**Example:**

```
/* enable xmc bank1_sram bank */
xmc_nor_sram_enable(XMC_BANK1_NOR_SRAM1, TRUE);
```

## 5.23.7 xmc\_ext\_timing\_config function

The table below describes the function xmc\_ext\_timing\_config.

**Table 506. xmc\_ext\_timing\_configfunction**

Name	Description
Function name	xmc_ext_timing_config
Function prototype	void xmc_ext_timing_config(xmc_nor_sram_subbank_type xmc_sub_bank, uint16_t w2w_timing, uint16_t r2r_timing);
Function description	Configure nor/sram extension controller
Input parameter 1	xmc_subbank: subbank selection
Input parameter 2	w2w_timing: bus turnaround phase duration between two consecutive write operations.
Input parameter 3	r2r_timing: bus turnaround phase duration between two consecutive read operations.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### xmc\_subbank

Select a subbank to reset.

XMC\_BANK1\_NOR\_SRAM1: xmc\_subbank1

XMC\_BANK1\_NOR\_SRAM2: xmc\_subbank2

XMC\_BANK1\_NOR\_SRAM3: xmc\_subbank3

XMC\_BANK1\_NOR\_SRAM4: xmc\_subbank4

### w2w\_timing

This is used to define the bus turnaround phase duration between two consecutive write operations.

00000000: 1 HCLK clock cycle inserted between two consecutive write operations

00000001: 2 HCLK clock cycles inserted between two consecutive write operations

.....

00001000: 9 HCLK clock cycles (default value) inserted between two consecutive read operations

.....

11111111: 256 HCLK clock cycles inserted between two consecutive read operations

### r2r\_timing

This is used to define the bus turnaround phase duration between two consecutive read operations.

00000000: 1 HCLK clock cycle inserted between two consecutive write operations

00000001: 2 HCLK clock cycles inserted between two consecutive write operations

.....

00001000: 9 HCLK clock cycles (default value) inserted between two consecutive read operations

.....

11111111: 256 HCLK clock cycles inserted between two consecutive write operations

### Example:

```
/* bus turnaround phase for consecutive read duration and consecutive write duration */
xmc_ext_timing_config(XMC_BANK1_NOR_SRAM1, 0x08, 0x08);
```

## 5.23.8 xmc\_nand\_reset function

The table below describes the function xmc\_nand\_reset.

**Table 507. xmc\_nand\_reset function**

Name	Description
Function name	xmc_nand_reset
Function prototype	void xmc_nand_reset(xmc_class_bank_type xmc_bank);
Function description	Reset nand controller
Input parameter	xmc_bank: define nand controller, it can be XMC_BANK2_NAND
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**Example:**

```
/* reset nand flash */
xmc_nand_reset(XMC_BANK2_NAND);
```

## 5.23.9 xmc\_nand\_init function

The table below describes the function xmc\_nand\_init.

**Table 508. xmc\_nand\_init function**

Name	Description
Function name	xmc_nand_init
Function prototype	void xmc_nand_init(xmc_nand_init_type* xmc_nand_init_struct);
Function description	Initialize nand controller
Input parameter	xmc_nand_init_struct: xmc_nand_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### xmc\_nand\_init\_type struct

xmc\_nand\_init\_type is defined in the at32f403\_xmc.h:

typedef struct

```
{
    xmc_class_bank_type      nand_bank;
    xmc_nand_wait_type      wait_enable;
    xmc_data_width_type     bus_type;
    xmc_ecc_enable_type     ecc_enable;
    xmc_ecc_pagesize_type   ecc_pagesize;
    uint32_t                 delay_time_cycle;
    uint32_t                 delay_time_ar;
}
```

} xmc\_nand\_init\_type;

### Nand\_bank

Select a nand\_bank to initialize.

XMC\_BANK2\_NAND: Initialize bank2 nand controller

XMC\_BANK3\_NAND: Initialize bank3 nand controller

#### **wait\_enable**

Wait enable bit. This is to enable the wait feature for the NAND Flash memory bank.

XMC\_WAIT\_OPERATION\_DISABLE: Disabled

XMC\_WAIT\_OPERATION\_ENABLE: Enabled

#### **bus\_type**

External memory data width. This is to define the width of the external NAND Flash data bus.

XMC\_BUSTYPE\_8\_BITS: 8-bit data bus

XMC\_BUSTYPE\_16\_BITS: 16-bit data bus

#### **ecc\_enable**

xmc data bus bit width.

XMC\_BUSTYPE\_8\_BITS: 8-bit data bus

XMC\_BUSTYPE\_16\_BITS: 16-bit data bus

#### **ecc\_pagesize**

ECC page size

000: 256 bytes

001: 512 bytes

010: 1024 bytes

011: 2048 bytes

100: 4096 bytes

101: 8192 bytes

#### **delay\_time\_cycle**

The delay time from CLE to RE, that is, from the CLE falling edge to the falling edge of RE.

0000:1 HCLK clock cycle

.....

1111:16 HCLK clock cycles

#### **delay\_time\_ar**

The delay time from ALE to RE, that is, from the ALE falling edge to the falling edge of RE.

0000:1 HCLK clock cycle

.....

1111:16 HCLK clock cycles

#### **Example:**

```
xmc_nand_init_type nand_init_struct;  
nand_init_struct.nand_bank = XMC_BANK2_NAND;  
nand_init_struct.wait_enable = XMC_WAIT_OPERATION_DISABLE;  
nand_init_struct.bus_type = XMC_BUSTYPE_8_BITS;  
nand_init_struct.ecc_enable = XMC_ECC_OPERATION_DISABLE;  
nand_init_struct.ecc_pagesize = XMC_ECC_PAGESIZE_2048_BYTES;  
nand_init_struct.delay_time_cycle = 0x10;  
nand_init_struct.delay_time_ar = 0x10;  
/* xmc nand flash configuration */  
xmc_nand_init(&nand_init_struct);
```

## 5.23.10 xmc\_nand\_timing\_config function

The table below describes the function xmc\_nand\_timing\_config.

**Table 509. xmc\_nand\_timing\_config function**

Name	Description
Function name	xmc_nand_timing_config
Function prototype	void xmc_nand_timing_config(xmc_nand_timinginit_type* xmc_common_spacetiming_struct, xmc_nand_timinginit_type* xmc_attribute_spacetiming_struct);
Function description	Configure nand controller timings
Input parameter 1	xmc_common_spacetiming_struct: xmc_nand_timinginit_type pointer, indicating common space timing parameters
Input parameter 2	xmc_attribute_spacetiming_struct: xmc_nand_timinginit_type pointer, indicating the space timing parameters in a special space
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### xmc\_nand\_timinginit\_type struct

xmc\_nand\_timinginit\_type is defined in the at32f403\_xmc.h:

```
typedef struct
{
    xmc_class_bank_type    class_bank;
    uint32_t                mem_setup_time;
    uint32_t                mem_waite_time;
    uint32_t                mem_hold_time;
    uint32_t                mem_hiz_time;
} xmc_nand_timinginit_type;
```

#### class\_bank

nand flash bank selection.

XMC\_BANK2\_NAND

XMC\_BANK3\_NAND

#### mem\_setup\_time

For NAND Flash write accesses to common memory space, this bit is used to define the number of HCLK clock cycles to set up the address.

00000000: 0 HCLK clock cycle inserted between two consecutive write operations

00000001: 1 HCLK clock cycle inserted between two consecutive write operations

.....  
11111111: 255 HCLK clock cycles inserted between two consecutive write operations

#### mem\_waite\_time

For NAND Flash write accesses to common memory space, this is used to define the number of HCLK clock cycles to be waited during which the XMC\_NWE and XMC\_NOE are active low.

00000000: 0 HCLK clock cycle inserted between two consecutive write operations

00000001: 1 HCLK clock cycle inserted between two consecutive write operations

.....

11111111: 255 HCLK clock cycles inserted between two consecutive write operations

#### **mem\_hold\_time**

For NAND Flash write accesses to the common memory space, this is used to define the number of HCLK clock cycles during which the data are held

00000000: Reserved

00000001: 1 HCLK clock cycle inserted between two consecutive write operations

.....

11111111: 255 HCLK clock cycles inserted between two consecutive write operations

#### **mem\_hiz\_time**

This is used to define the number of HCLK clock cycles during which the data bus is kept in Hi-Z after the start of a NAND Flash write access to common memory space.

00000000: 0 HCLK clock cycle inserted between two consecutive write operations

00000001: 1 HCLK clock cycle inserted between two consecutive write operations

.....

11111111: 255 HCLK clock cycles inserted between two consecutive write operations

#### **Example:**

```
xmc_regular_spacetimingstruct.class_bank = XMC_BANK2_NAND;
xmc_regular_spacetimingstruct.mem_setup_time = 254;
xmc_regular_spacetimingstruct.mem_hiz_time = 254;
xmc_regular_spacetimingstruct.mem_hold_time = 254;
xmc_regular_spacetimingstruct.mem_wait_time = 254;
xmc_nand_timing_config(&xmc_regular_spacetimingstruct, &xmc_regular_spacetimingstruct);
```

### **5.23.11 xmc\_nand\_default\_para\_init function**

The table below describes the function xmc\_nand\_default\_para\_init.

**Table 510. xmc\_nand\_default\_para\_init**

Name	Description
Function name	xmc_nand_default_para_init
Function prototype	void xmc_nand_default_para_init(xmc_nand_init_type* xmc_nand_init_struct);
Function description	Initialize the parameters of the xmc_nand_init_struct
Input parameter	xmc_nand_init_struct: xmc_nand_init_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below shows the default values of members in the xmc\_nand\_init\_struct.

**Table 511. xmc\_nand\_init\_struct**

Member	Default values
nand_bank	XMC_BANK2_NAND
wait_enable	XMC_WAIT_OPERATION_DISABLE
bus_type	XMC_BUSTYPE_8_BITS
ecc_enable	XMC_ECC_OPERATION_DISABLE
ecc_pagesize	XMC_ECC_PAGESIZE_256_BYTES
delay_time_cycle	0x0
delay_time_ar	0x0

**Example:**

```
/* fill each nand_init_struct member with its default value */
xmc_nand_default_para_init(&nand_init_struct);
```

### 5.23.12 xmc\_nand\_timing\_default\_para\_init function

The table below describes the function xmc\_nand\_timing\_default\_para\_init.

**Table 512. xmc\_nand\_timing\_default\_para\_init function**

Name	Description
Function name	xmc_nand_timing_default_para_init
Function prototype	void xmc_nand_timing_default_para_init(xmc_nand_timinginit_type* xmc_common_spacetiming_struct, xmc_nand_timinginit_type* xmc_attribute_spacetiming_struct);
Function description	Initialize the parameters of the xmc_common_spacetiming_struct and xmc_attribute_spacetiming_struct
Input parameter	xmc_common_spacetiming_struct: xmc_nand_timinginit_type pointer xmc_attribute_spacetiming_struct: mc_nand_timinginit_type pointer
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below shows the default values of members in the xmc\_rw\_timing\_struct

xmc\_common\_spacetiming\_struct and xmc\_w\_timing\_struct

xmc\_attribute\_spacetiming\_struct

**Table 513. xmc\_common\_spacetiming\_struct and xmc\_attribute\_spacetiming\_struct**

Member	Default values
class_bank	XMC_BANK2_NAND
mem_hold_time	0xFC
mem_wait_time	0xFC
mem_setup_time	0xFC
mem_hiz_time	0xFC

**Example:**

```
xmc_nand_timinginit_type xmc_regular_spacetimingstruct;
/* fill each xmc_regular_spacetimingstruct member with its default value */
xmc_nand_timing_default_para_init(&xmc_regular_spacetimingstruct, &xmc_regular_spacetimingstruct);
```

### 5.23.13 xmc\_nand\_enable function

The table below describes the function xmc\_nand\_enable.

**Table 514. xmc\_nand\_enable function**

Name	Description
Function name	xmc_nand_enable
Function prototype	void xmc_nand_enable(xmc_class_bank_type xmc_bank, confirm_state new_state);
Function description	Enable nand flash controller
Input parameter 1	xmc_bank: bank selection
Input parameter 2	new_state: enable or disable
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**xmc\_bank**

Bank selection

XMC\_BANK2\_NAND

XMC\_BANK3\_NAND

**new\_state**

FALSE: Disabled

TRUE: Enabled

**Example:**

```
/* xmc nand bank enable*/
xmc_nand_enable(XMC_BANK2_NAND, TRUE);
```

### 5.23.14 xmc\_nand\_ecc\_enable function

The table below describes the function xmc\_nand\_ecc\_enable.

**Table 515. xmc\_nand\_ecc\_enable function**

Name	Description
Function name	xmc_nand_ecc_enable
Function prototype	void xmc_nand_ecc_enable(xmc_class_bank_type xmc_bank, confirm_state new_state);
Function description	Enable nand flash controller ECC feature
Input parameter 1	xmc_bank: bank selection
Input parameter 2	new_state: enable or disable ECC feature.
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**xmc\_bank**

Select a bank to ECC enable.

XMC\_BANK2\_NAND

XMC\_BANK3\_NAND

**new\_state**

FALSE: Disable ECC feature

TRUE: Enable ECC feature

**Example:**

```
/* calculate ecc value while transmitting */
xmc_nand_ecc_enable(XMC_BANK2_NAND, TRUE);
```

## 5.23.15 xmc\_ecc\_get function

The table below describes the function xmc\_ecc\_get.

**Table 516. xmc\_ecc\_get function**

Name	Description
Function name	xmc_ecc_get
Function prototype	uint32_t xmc_ecc_get(xmc_class_bank_type xmc_bank);
Function description	Get nand flash ECC value
Input parameter 1	xmc_bank: bank selection
Output parameter	NA
Return value	ECC value
Required preconditions	NA
Called functions	NA

### xmc\_bank

Select a bank to enable ECC.

XMC\_BANK2\_NAND

XMC\_BANK3\_NAND

### Example:

```
/* get ecc value */
xmc_ecc_get(XMC_BANK2_NAND, TRUE);
```

## 5.23.16 xmc\_interrupt\_enable function

The table below describes the function xmc\_interrupt\_enable.

**Table 517. xmc\_interrupt\_enable function**

Name	Description
Function name	xmc_interrupt_enable
Function prototype	void xmc_interrupt_enable(xmc_class_bank_type xmc_bank, xmc_interrupt_sources_type xmc_int, confirm_state new_state);
Function description	Enable xmc interrupts
Input parameter 1	xmc_bank: bank selection
Input parameter 2	xmc_int: interrupt selection
Input parameter 3	new_state: enable or disable interrupts
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### xmc\_bank

Bank selection.

XMC\_BANK2\_NAND

XMC\_BANK3\_NAND

### xmc\_int

XMC\_INT\_RISING\_EDGE: XMC rising edge detection interrupt

XMC\_INT\_LEVEL: XMC high level detection interrupt

XMC\_INT\_FALLING\_EDGE: XMC falling edge detection interrupt

**new\_state**

FALSE: Interrupt disabled

TRUE: Interrupt enabled

**Example:**

```
/* xmc rising edge detection interrupt enable */
xmc_interrupt_enable(XMC_BANK2_NAND, XMC_INT_RISING_EDGE, TRUE);
```

### 5.23.17 xmc\_flag\_status\_get function

The table below describes the function xmc\_flag\_status\_get.

**Table 518. xmc\_flag\_status\_get function**

Name	Description
Function name	xmc_flag_status_get
Function prototype	flag_status xmc_flag_status_get(xmc_class_bank_type xmc_bank, xmc_interrupt_flag_type xmc_flag);
Function description	Get XMC flag status
Input parameter 1	xmc_bank: xmc bank selection
Input parameter 2	xmc_flag: flag selection
Output parameter	NA
Return value	flag_status: check if the selected flag is set or not.
Required preconditions	NA
Called functions	NA

**xmc\_bank**

XMC bank selection.

**xmc\_flag**

xmc\_flag is used for flag selection to get its status, including:

XMC\_RISINGEDGE\_FLAG: Rising edge status

XMC\_LEVEL\_FLAG: High level status

XMC\_FALLINGEDGE\_FLAG: Falling edge status

XMC\_FEMPT\_FLAG: FIFO empty status

**flag\_status**

RESET: Flag is not set

SET: Flag is set

**Example:**

```
if(xmc_flag_status_get (XMC_BANK2_NAND, XMC_RISINGEDGE_FLAG) != RESET)
{
    .....
}
```

## 5.23.18 xmc\_interrupt\_flag\_clear function

The table below describes the function xmc\_interrupt\_flag\_clear.

**Table 519. xmc\_interrupt\_flag\_clear function**

Name	Description
Function name	xmc_interrupt_flag_status_get
Function prototype	flag_status xmc_interrupt_flag_status_get(xmc_class_bank_type xmc_bank, xmc_interrupt_flag_type xmc_flag)
Function description	Get XMC interrupt flag status
Input parameter 1	xmc_bank: xmc bank selection
Input parameter 2	xmc_flag: flag selection
Output parameter	NA
Return value	flag_status: check if the selected flag is set or not.
Required preconditions	NA
Called functions	NA

### xmc\_bank

XMC bank selection.

### xmc\_flag

xmc\_flag is used for flag selection to get its status, including:

XMC\_RISINGEDGE\_FLAG: Rising edge status

XMC\_LEVEL\_FLAG: High level status

XMC\_FALLINGEDGE\_FLAG: Falling edge status

### flag\_status

RESET: Flag is not set

SET: Flag is set

### Example

```
if(xmc_interrupt_flag_status_get (XMC_BANK2_NAND, XMC_RISINGEDGE_FLAG) != RESET)
{
    .....
}
```

### 5.23.19 xmc\_flag\_clear function

The table below describes the function xmc\_flag\_clear.

**Table 520. xmc\_flag\_clear function**

Name	Description
Function name	xmc_flag_clear
Function prototype	void xmc_flag_clear(xmc_class_bank_type xmc_bank, xmc_interrupt_flag_type xmc_flag);
Function description	Clear flag
Input parameter 1	xmc_bank: xmc bank selection
Input parameter 2	xmc_flag: flag selection
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### xmc\_bank

XMC bank selection.

#### xmc\_flag

xmc\_flag is used for flag selection to get its status, including:

XMC_RISINGEDGE_FLAG:	Rising edge status
XMC_LEVEL_FLAG:	High level status
XMC_FALLINGEDGE_FLAG:	Falling edge status
XMC_FEMPT_FLAG:	FIFO empty status

#### Example:

```
if(xmc_flag_status_get(XMC_BANK2_NAND, XMC_RISINGEDGE_FLAG) != RESET)
{
    .....
    xmc_flag_clear(XMC_BANK2_NAND, XMC_RISINGEDGE_FLAG);
}
```

### 5.23.20 xmc\_pccard\_reset function

The table below describes the function xmc\_pccard\_reset.

**Table 521. xmc\_pccard\_reset function**

Name	Description
Function name	xmc_pccard_reset
Function prototype	void xmc_pccard_reset(void);
Function description	Reset pccard controller
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

#### Example:

```
/* reset pccard */
xmc_pccard_reset();
```

## 5.23.21 xmc\_pccard\_init function

The table below describes the function xmc\_pccard\_init.

**Table 522. xmc\_pccard\_init function**

Name	Description
Function name	xmc_pccard_init
Function prototype	void xmc_pccard_init(xmc_pccard_init_type* xmc_pccard_init_struct);
Function description	Initialize pccard controller
Input parameter	NA
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### xmc\_pccard\_init\_type struct

xmc\_pccard\_init\_type is defined in the at32f403\_xmc.h

typedef struct

```
{
    xmc_nand_pccard_wait_type    enable_wait;
    uint32_t                      delay_time_cr;
    uint32_t                      delay_time_ar;
} xmc_pccard_init_type;
```

#### enable\_wait

Enable PC card memory block wait.

XMC\_WAIT\_OPERATION\_DISABLE: Disable

XMC\_WAIT\_OPERATION\_ENABLE: Enable

#### delay\_time\_cr

Delay time from CLE falling edge to RE falling edge

0000: 1 HCLK cycle

.....

1111: 16 HCLK cycles

#### delay\_time\_ar

Delay time from ALE falling edge to RE falling edge

0000: 1 HCLK cycle

.....

1111: 16 HCLK cycles

#### Example:

```
xmc_pccard_init_type xmc_pccard_init_struct;
xmc_pccard_init_struct.enable_wait = XMC_WAIT_OPERATION_DISABLE;
xmc_pccard_init_struct.delay_time_cr = 0xF;
xmc_pccard_init_struct.delay_time_ar = 0xF;
/* xmc pccard configuration */
xmc_pccard_init (&xmc_pccard_init_struct);
```

## 5.23.22 xmc\_pccard\_timing function

The table below describes the function xmc\_pccard\_timing.

**Table 523. xmc\_pccard\_timing function**

Name	Description
Function name	xmc_pccard_timing_config
Function prototype	void xmc_pccard_timing_config(xmc_nand_pccard_timinginit_type* xmc_common_spacetiming_struct, xmc_nand_pccard_timinginit_type* xmc_attribute_spacetiming_struct, xmc_nand_pccard_timinginit_type* xmc_iospace_timing_struct);
Function description	Set pccard controller timing
Input parameter 1	xmc_common_spacetiming_struct: pointer to the structure xmc_nand_pccard_timinginit_type (common space timing parameters)
Input parameter 2	xmc_attribute_spacetiming_struct: pointer to the structure xmc_nand_pccard_timinginit_type (special space timing parameters)
Input parameter 3	xmc_iospace_timing_struct: pointer to the structure xmc_nand_pccard_timinginit_type (special space timing parameters)
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

### xmc\_nand\_pccard\_timinginit\_type struct

xmc\_nand\_pccard\_timinginit\_type is defined in the at32f403\_xmc.h

typedef struct

```
{
    xmc_class_bank_type      class_bank;
    uint32_t                 mem_setup_time;
    uint32_t                 mem_waite_time;
    uint32_t                 mem_hold_time;
    uint32_t                 mem_hiz_time;
} xmc_nand_pccard_timinginit_type;
```

#### class\_bank

Select NAND Flash bank.

XMC\_BANK4\_PCCARD

#### mem\_setup\_time

Setup time of the address line when accessing common memory

00000000: insert extra 0 HCLK cycle

00000001: insert extra 1 HCLK cycle

.....

11111111: insert extra 255 HCLKs

#### mem\_waite\_time

Wait time from which the XMC\_NWE and XMC\_NOE become low when accessing common memory

00000000: insert extra 0 HCLK cycle  
00000001: insert extra 1 HCLK cycle

.....

11111111: insert extra 255 HCLKs

#### **mem\_hold\_time**

Data bus hold time when accessing common memory.

00000000: Reserved

00000001: insert extra 1 HCLK cycle

.....

11111111: insert extra 255 HCLK cycles

#### **mem\_hiz\_time**

Data bus in high resistance time when writing operationg to pccard in common memory

00000000: insert extra 0 HCLK

00000001: insert extra 1 HCLK cycle

.....

11111111: insert extra 255 HCLK cycles

#### **Example:**

```
xmc_nand_pccard_timinginit_type xmc_common_spacetiming_struct, xmc_attribute_spacetiming_struct,  
xmc_iospace_timing_struct;  
xmc_common_spacetiming_struct.class_bank = XMC_BANK4_PCCARD;  
xmc_common_spacetiming_struct.mem_setup_time = 254;  
xmc_common_spacetiming_struct.mem_hiz_time = 254;  
xmc_common_spacetiming_struct.mem_hold_time = 254;  
xmc_common_spacetiming_struct.mem_waite_time = 254;  
  
xmc_attribute_spacetiming_struct.class_bank = XMC_BANK4_PCCARD;  
xmc_attribute_spacetiming_struct.mem_setup_time = 254;  
xmc_attribute_spacetiming_struct.mem_hiz_time = 254;  
xmc_attribute_spacetiming_struct.mem_hold_time = 254;  
xmc_attribute_spacetiming_struct.mem_waite_time = 254;  
  
xmc_iospace_timing_struct.class_bank = XMC_BANK4_PCCARD;  
xmc_iospace_timing_struct.mem_setup_time = 254;  
xmc_iospace_timing_struct.mem_hiz_time = 254;  
xmc_iospace_timing_struct.mem_hold_time = 254;  
xmc_iospace_timing_struct.mem_waite_time = 254;  
  
xmc_pccard_timing_config (&xmc_common_spacetiming_struct, &xmc_attribute_spacetiming_struct,&  
xmc_iospace_timing_struct);
```

### 5.23.23 xmc\_pccard\_default\_para\_init function

The table below describes the function xmc\_pccard\_default\_para\_init.

**Table 524. xmc\_pccard\_default\_para\_init function**

Name	Description
Function name	xmc_pccard_default_para_init
Function prototype	void xmc_pccard_default_para_init(xmc_pccard_init_type* xmc_pccard_init_struct);
Function description	Initialize the default parameters in the xmc_pccard_init_struct
Input parameter 1	xmc_pccard_init_struct: pointer to the structure xmc_pccard_init_type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The default values of the structure xmc\_nand\_init\_struct is shown in the table below.

**Table 525. xmc\_pccard\_init\_struct default value**

Parameter	Default value
enable_wait	XMC_WAIT_OPERATION_DISABLE
delay_time_ar	0x0
delay_time_cr	0x0

**Example:**

```
/* fill each xmc_pccard_init_struct member with its default value */  
xmc_pccard_default_para_init (&xmc_pccard_init_struct);
```

## 5.23.24 xmc\_pccard\_timing\_default\_para\_init function

The table below describes the function xmc\_pccard\_timing\_default\_para\_init.

**Table 526. xmc\_pccard\_timing\_default\_para\_init function**

Name	Description
Function name	xmc_pccard_timing_default_para_init
Function prototype	void xmc_pccard_timing_default_para_init(xmc_nand_pccard_timinginit_type* xmc_common_spacetiming_struct, xmc_nand_pccard_timinginit_type* xmc_attribute_spacetiming_struct, xmc_nand_pccard_timinginit_type* xmc_iospace_timing_struct);
Function description	Initialize the parameters of the xmc_common_spacetiming_struct, xmc_attribute_spacetiming_struct and xmc_iospace_timing_struct
Input parameter 1	xmc_common_spacetiming_struct: pointer to the xmc_nand_pccard_timinginit_type
Input parameter 2	xmc_attribute_spacetiming_struct: pointer to the xmc_nand_pccard_timinginit_type
Input parameter 3	xmc_iospace_timing_struct: pointer to the xmc_nand_pccard_timinginit_type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

The table below presents the default values of the xmc\_common\_spacetiming\_struct, xmc\_attribute\_spacetiming\_struct and xmc\_iospace\_timing\_struct

**Table 527. xmc\_common\_spacetiming\_struct and xmc\_attribute\_spacetiming\_struct**

Parameter	Default value
class_bank	XMC_BANK4_PCCARD
mem_hold_time	0xFC
mem_waite_time	0xFC
mem_setup_time	0xFC
mem_hiz_time	0xFC

### Example:

```
xmc_pccard_timinginit_type xmc_regular_spacetimingstruct;
/* fill each xmc_regular_spacetimingstruct member with its default value */
xmc_pccard_timing_default_para_init (&xmc_regular_spacetimingstruct, &xmc_regular_spacetimingstruct, &
xmc_regular_spacetimingstruct);
```

## 5.23.25 xmc\_pccard\_enable function

The table below describes the function xmc\_pccard\_enable.

**Table 528. xmc\_pccard\_enable function**

Name	Description
Function name	xmc_pccard_enable
Function prototype	void xmc_pccard_enable(confirm_state new_state);
Function description	Enable pccard controller
Input parameter 1	new_state: enable or disable controller
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

**new\_state**

FALSE: Disable controller

TRUE: Enable controller

**Example:**

```
/* xmc pccard bank enable*/
xmc_pccard_enable(TRUE);
```

## 6 Precautions

### 6.1 Device model replacement

While replacing the device model in an existing project or demo with another one, if necessary, the macro definition is also required to be changed with the device model. Therefore, it is necessary to check and find out the macro definition corresponding to the device model defined in [Table 1](#) before replacing the device model. The subsequent sections give a detailed description of how to replace a device model in KEIL and IAR environment (Just taking the at32f403avgt7 as an example as other devices share similar operations).

There are two steps to get this happen:

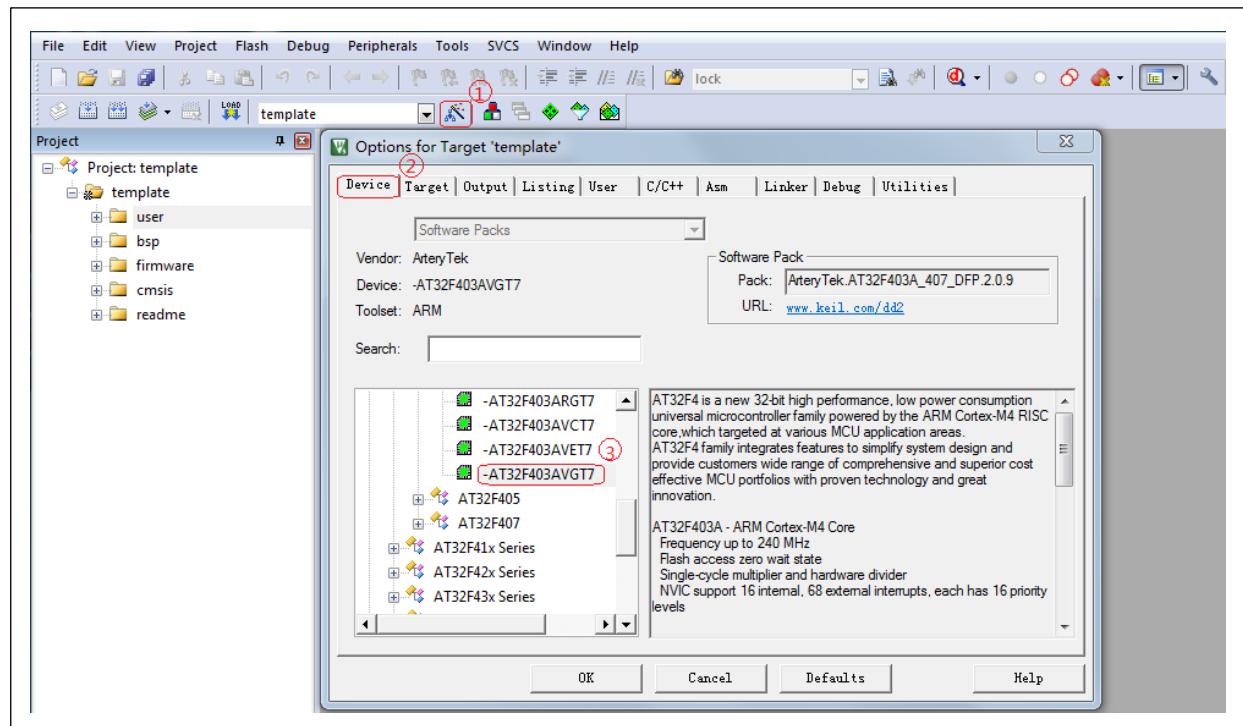
1. By changing device
2. By changing macro definition

#### 6.1.1 KEIL environment

Follow the steps and illustration below for device replacement in Keil environment:

- ① Click on magic stick “Options for Target”
- ② Click on “Device”
- ③ Select the desired device part number

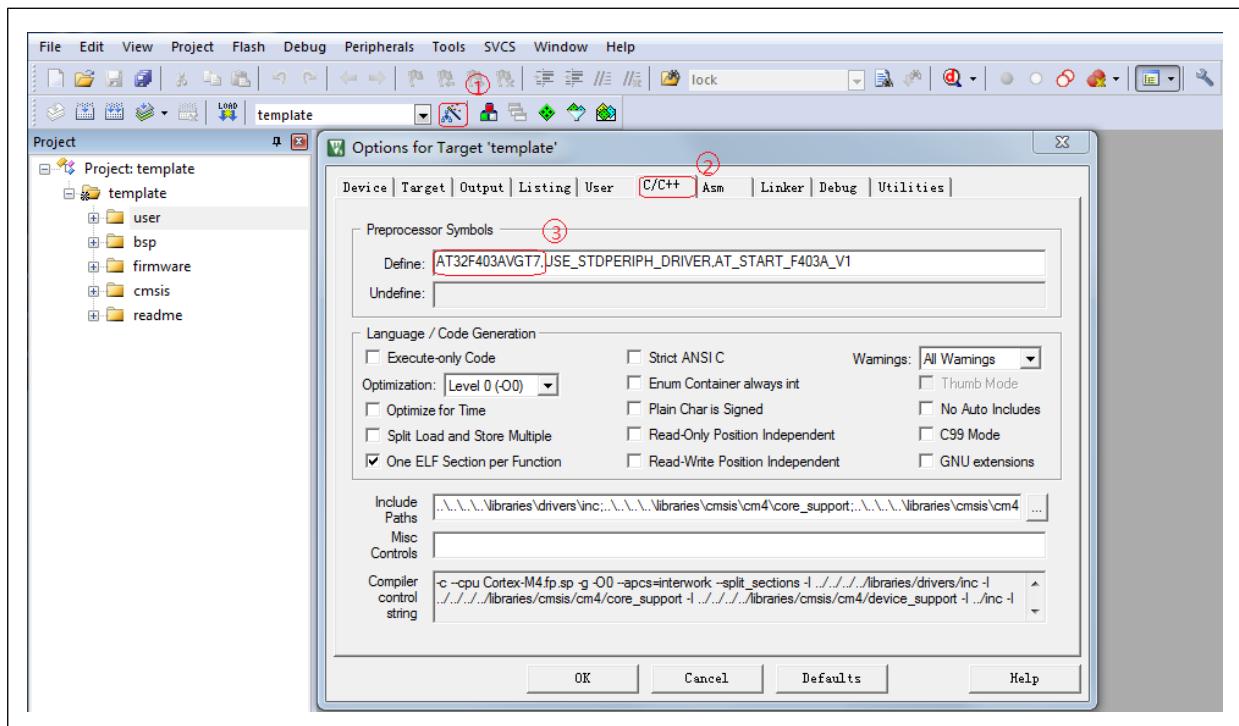
**Figure 29. Change device part number in Keil**



Follow the steps and illustration below for changing macro definition.

- ① Click on magic stick “Options for Target”.
- ② Click on “C/C++”.
- ③ Delete the original macro definition in “Define” box, and write the desired macro definition corresponding to the selected device part number based on [Table 1](#).

Figure 30. Change macro definition in Keil

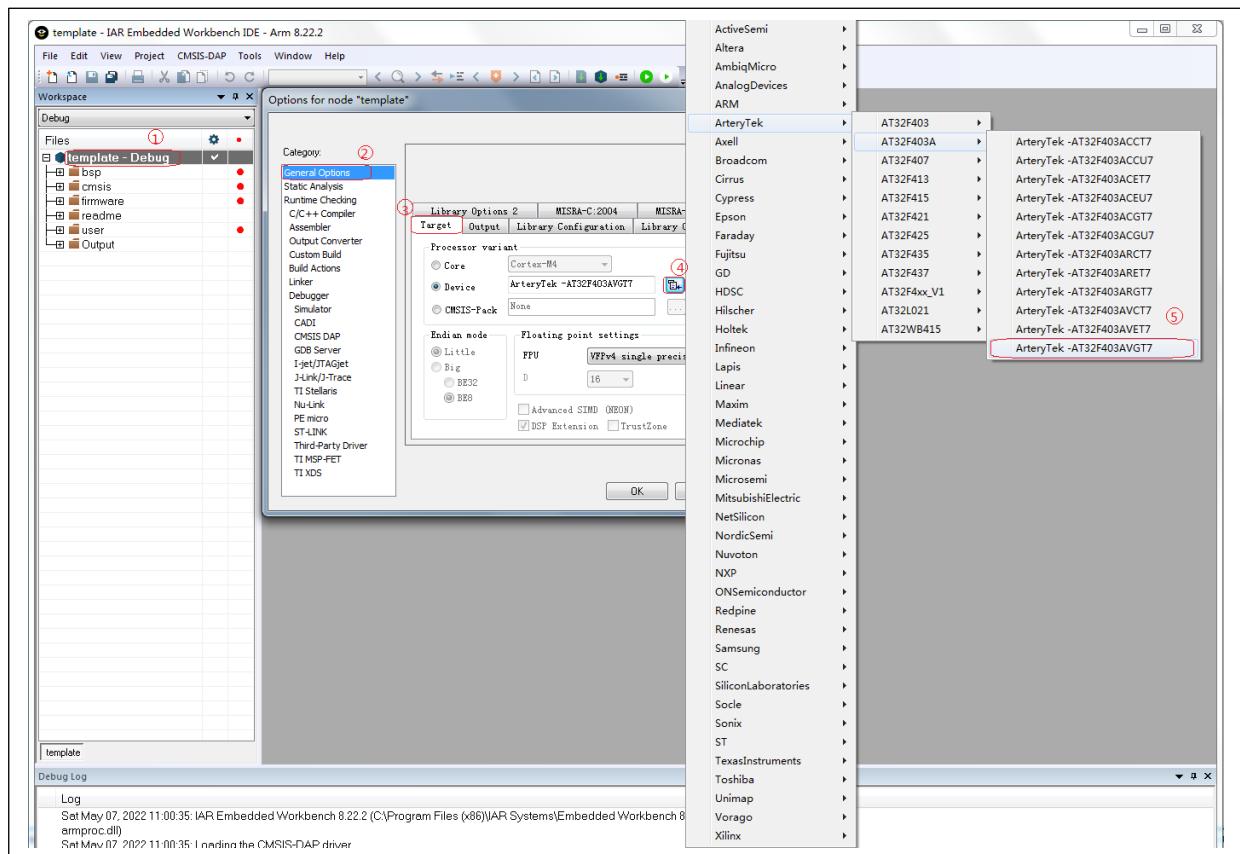


## 6.1.2 IAR environment

Follow the steps and illustration below for device replacement in IAR environment.

- ① Right click on the file name, and select “Options...”
- ② Select “General Options”
- ③ Select “Target”
- ④ Click on check box
- ⑤ Select the desired device part number.

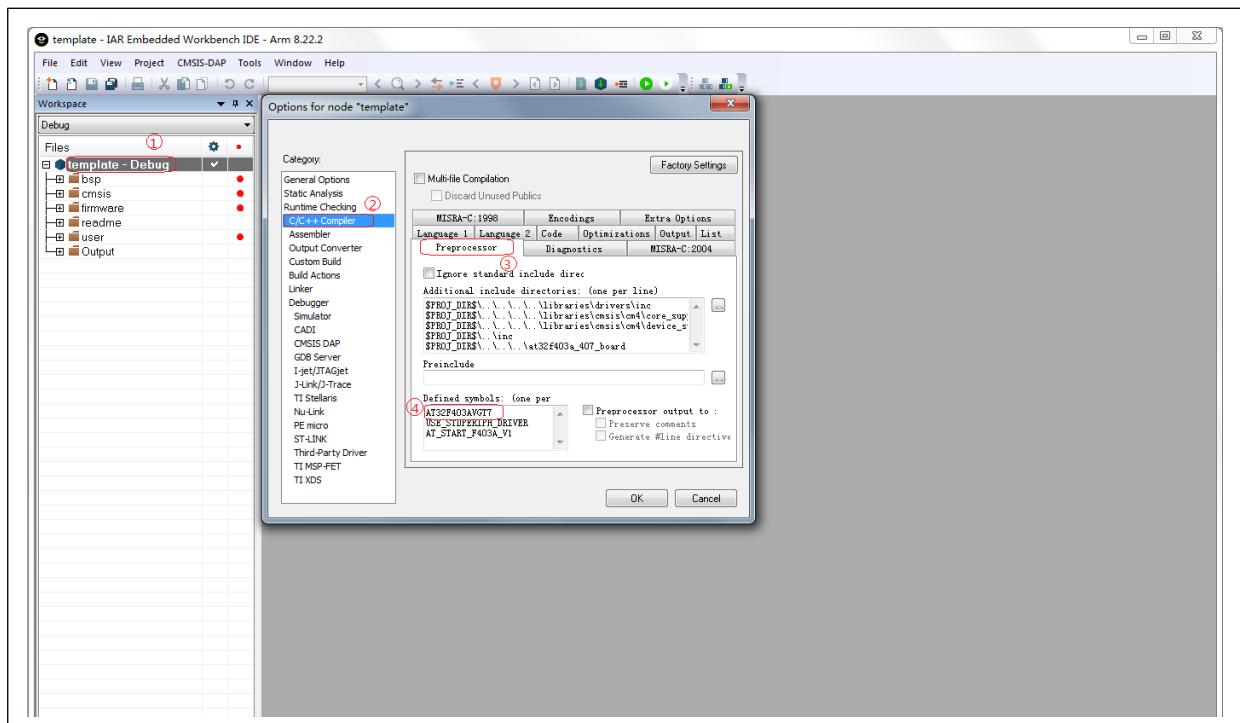
Figure 31. Change device part number in IAR



Follow the steps and illustration below for changing macro definition in IAR environment.

- ① Right click on the file name, and select “Options...”
- ② Select “C/C++ Compiler”
- ③ Click on “Preprocessor”
- ④ Delete the original macro definition in “Defined symbols” column, and write the desired macro definition corresponding to the selected device part number based on [Table 1](#).

Figure 32. Change macro definition in IAR



## 6.2 Failed to identify IC by JLink software in Keil

In special circumstances, the Keil project compiled by an engineer is unknown to the J-Link software even if it can be compiled by other engineers and identified by ICP software. For example, some warnings like below will be displayed.

Figure 33. Error warning 1

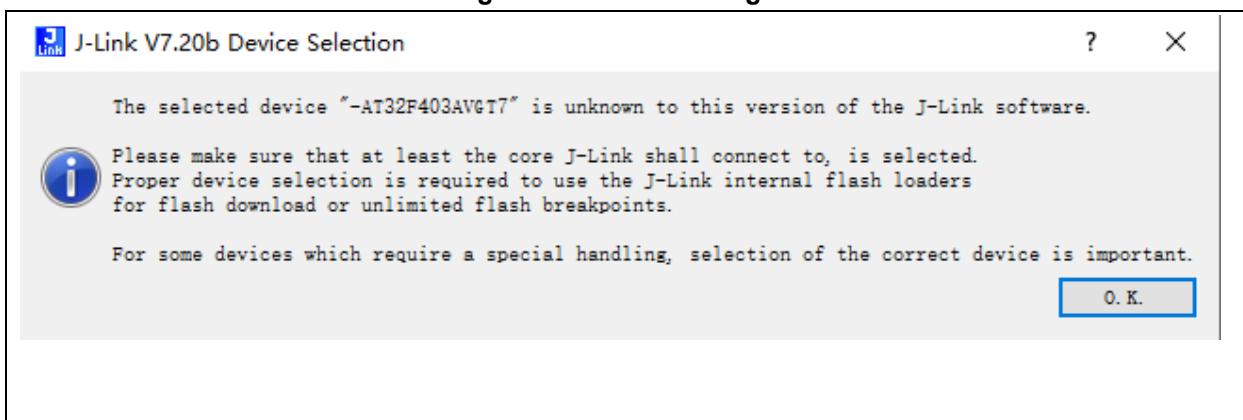


Figure 34. Error warning 2

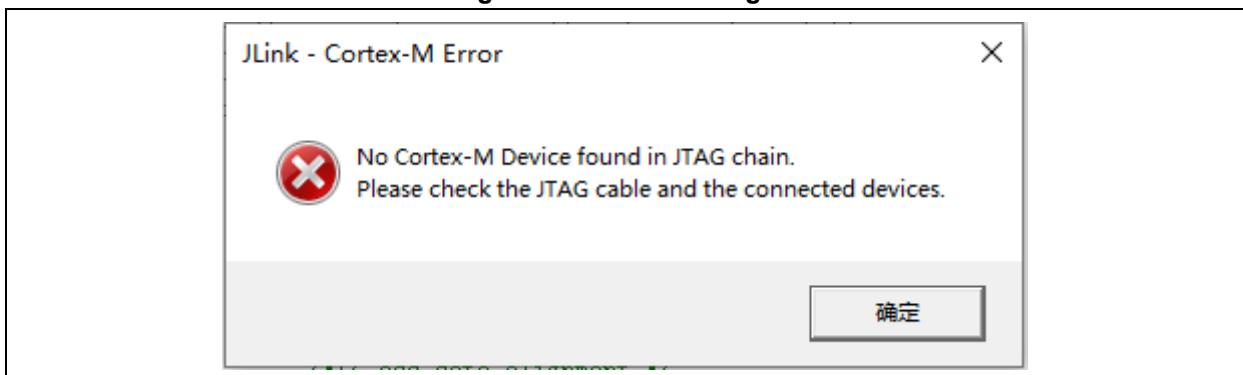
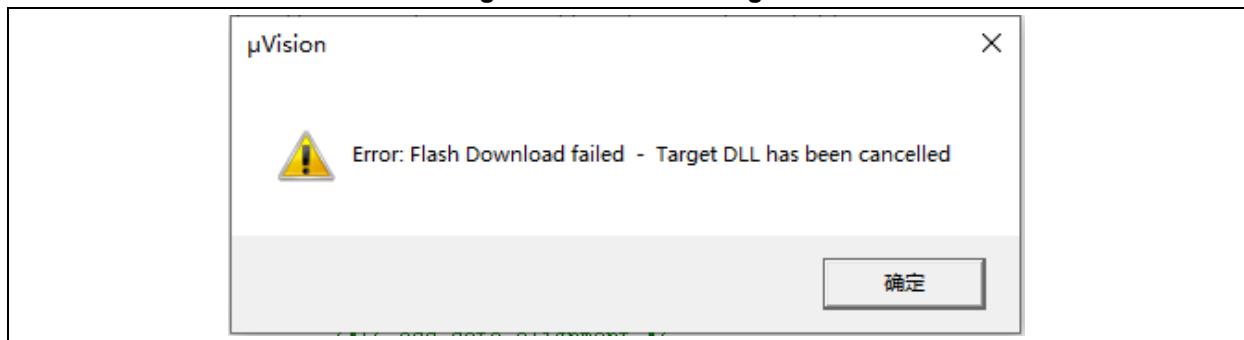


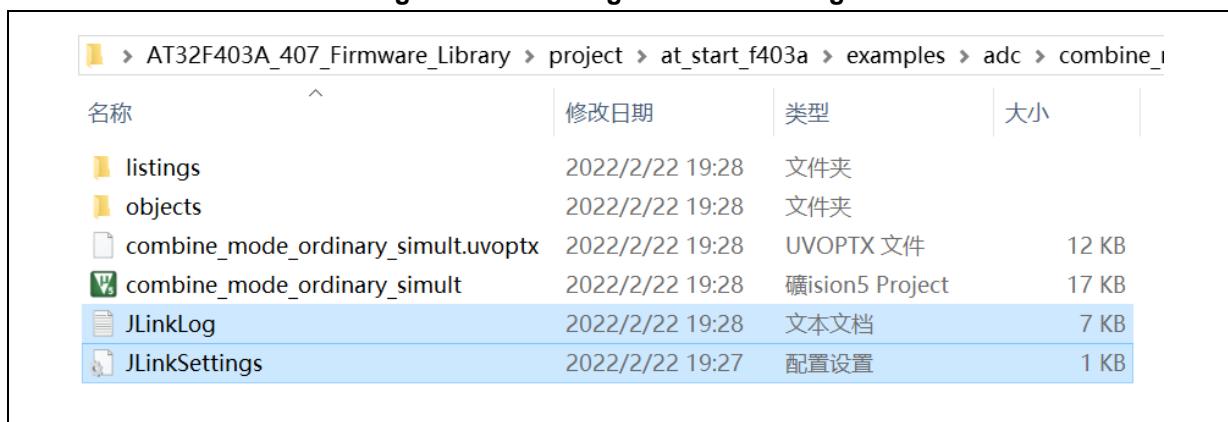
Figure 35. Error warning 3



### How to solve this problem?

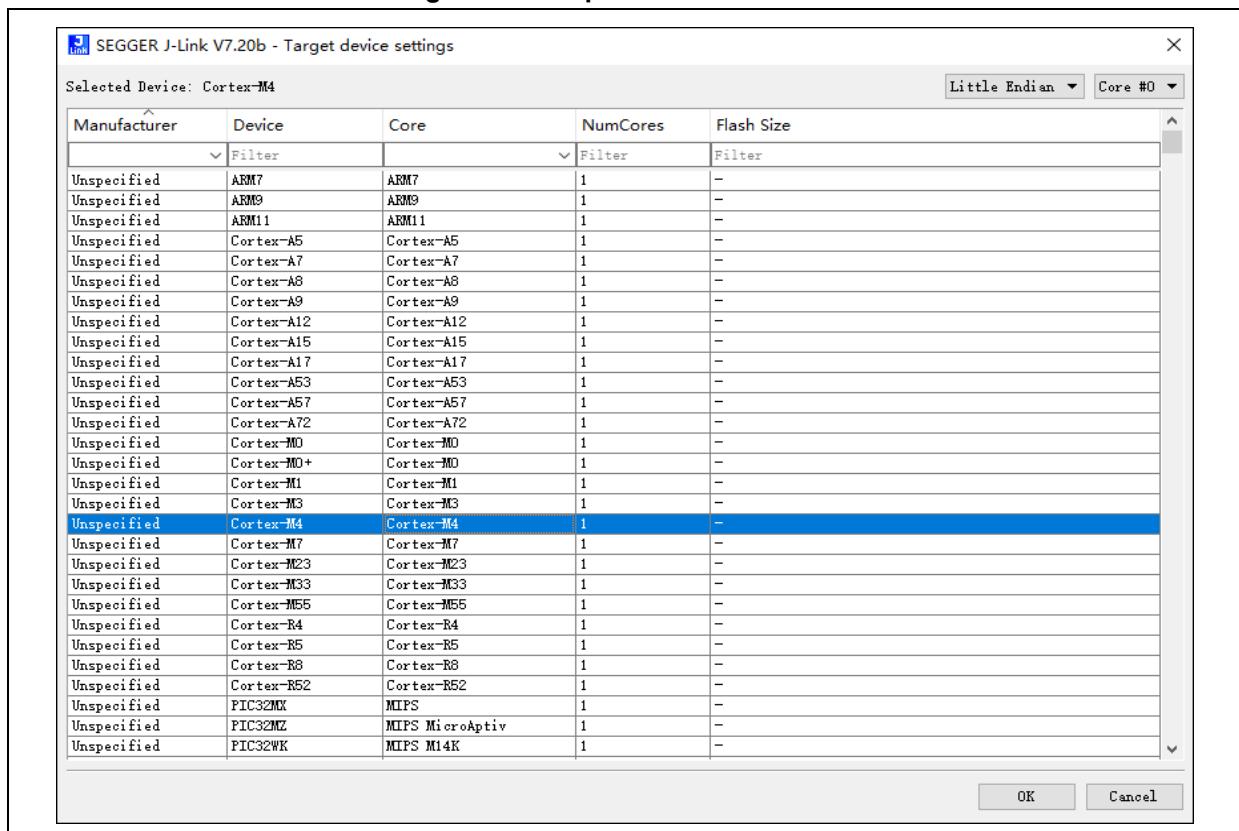
Step 1: Find “JLinkLog” and “JLinkSettings” files according to project path, and delete them.

Figure 36. JLinkLog and JLinkSettings



Step 2: Click on magic wand, go to “Debug”, select “Unspecified Cortex-M4”

Figure 37. Unspecified Cortex-M4



## 6.3 How to change HEXT crystal

All examples used in BSP implements frequency multiplication based on 8 MHz external high-speed crystal oscillator on the evaluation board. If a non-8 MHz external crystal is used in actual scenarios, it is necessary to modify clock configuration in BSP to allow for accurate and stable clock frequency.

Therefore, the “AT32\_New\_Clock\_Configuration” tool is specially developed by Artery to generate the desired BSP system clock code file, including external clock source, frequency division factor, frequency multiplication factor, clock source selection and other parameters, marked in red in Figure 38. After the completion of parameter configuration, it is ready to generate code file, avoiding complicated operations involved in code modification.

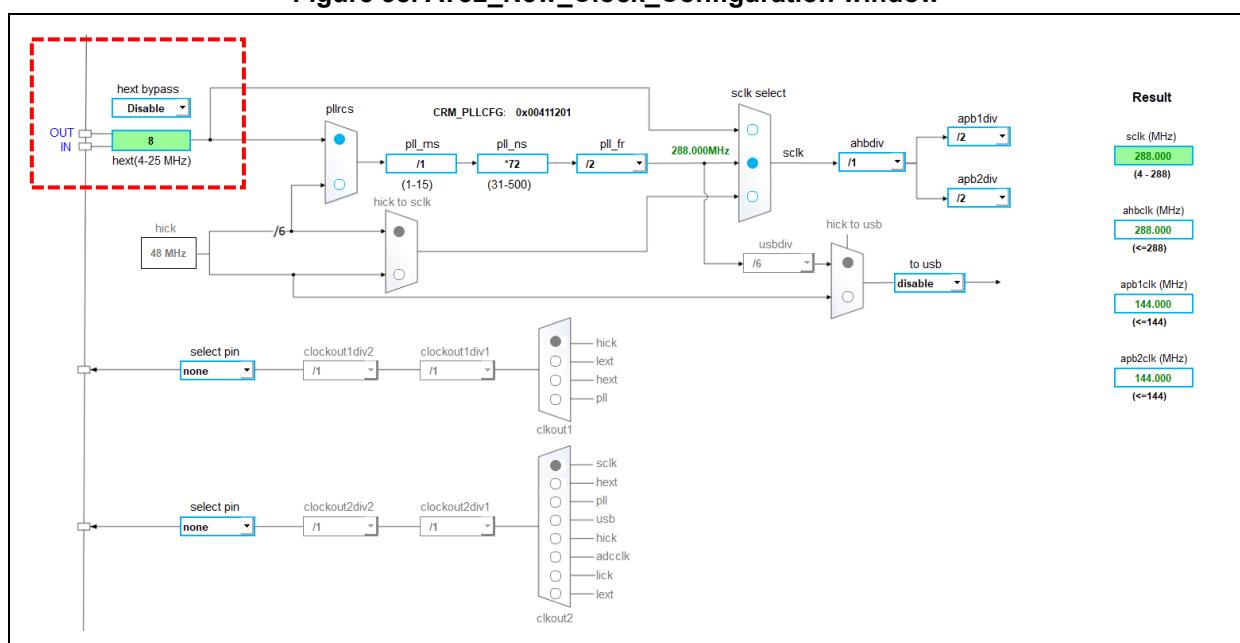
The users simply need to replace the original one in BSP demo with the newly generated clock code file (at32f4xx\_clock.c/ at32f4xx\_clock.h/ at32f4xx\_conf.h), and call the function system\_clock\_config in main function.

Also, it is necessary to replace the macro definition HEXT\_VALUE in the at32f4xx\_conf.h. Taking the AT32F403A as an example, the HEXT\_VALUE of the at32f403a\_407\_conf.h is defined as:

```
#define HEXT_VALUE ((uint32_t)8000000) /*!< value of the high speed external crystal in hz */
```

Figure 38 shows the window of AT32\_New\_Clock\_Configuration tool.

**Figure 38. AT32\_New\_Clock\_Configuration window**



For more information on the AT32\_New\_Clock\_Configuration, please refer to the corresponding Application Note shown in Table 529, which are all available from the official website of Artery.

**Table 529. Clock configuration guideline**

Part number	Application note
AT32F403A/407 clock configuration	AN0082
AT32F435/437 clock configuration	AN0084
AT32F421 clock configuration	AN0116
AT32F415 clock configuration	AN0117
AT32F413 clock configuration	AN0118
AT32F425 clock configuration	AN0121

## 7 Revision history

Table 530. Document revision history

Date	Revision	Changes
2022.02.11	2.0.0	Initial release
2022.05.09	2.0.1	Added the <a href="#">Section 6.1 Device model replacement</a>
2022.06.15	2.0.2	Added the descriptions of peripherals functions.
2022.11.15	2.0.3	Modified abbreviations of peripherals in I2C section.
2023.07.18	2.0.4	Added CRC related registers and functions.
2023.10.26	2.0.5	Added interrupt_flag_get function for each IP.
2024.03.01	2.0.6	Changed “PWC” to “NVIC” in NVIC section.

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

Purchasers are solely responsible for the selection and use of ARTERY's products and services, and ARTERY assumes no liability whatsoever relating to the choice, selection or use of the ARTERY products and services described herein.

No license, express or implied, to any intellectual property rights is granted under this document. If any part of this document deals with any third party products or services, it shall not be deemed a license grant by ARTERY for the use of such third party products or services, or any intellectual property contained therein, or considered as a warranty regarding the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

Unless otherwise specified in ARTERY's terms and conditions of sale, ARTERY provides no warranties, express or implied, regarding the use and/or sale of ARTERY products, including but not limited to any implied warranties of merchantability, fitness for a particular purpose (and their equivalents under the laws of any jurisdiction), or infringement of any patent, copyright or other intellectual property right.

Purchasers hereby agree that ARTERY's products are not designed or authorized for use in: (A) any application with special requirements of safety such as life support and active implantable device, or system with functional safety requirements; (B) any aircraft application; (C) any aerospace application or environment; (D) any weapon application, and/or (E) or other uses where the failure of the device or product could result in personal injury, death, property damage. Purchasers' unauthorized use of them in the aforementioned applications, even if with a written notice, is solely at purchasers' risk, and Purchasers are solely responsible for meeting all legal and regulatory requirements in such use.

Resale of ARTERY products with provisions different from the statements and/or technical features stated in this document shall immediately void any warranty grant by ARTERY for ARTERY products or services described herein and shall not create or expand in any manner whatsoever, any liability of ARTERY.