

AT32F425 Firmware BSP&Pack

Introduction

This application note is written to give a brief description of how to use AT32F425 BSP (Board Support Package) and install AT32 pack.

Contents

1	Overview	36
2	How to install Pack	37
2.1	IAR Pack installation.....	37
2.2	Keil_v5 Pack installation.....	39
2.3	Keil_v4 Pack installation.....	41
2.4	Segger Pack installation.....	43
3	Flash algorithm file	47
3.1	How to use Keil algorithm file	47
3.2	How to use IAR algorithm files.....	49
4	BSP introduction.....	53
4.1	Quick start	53
4.1.1	Template project.....	53
4.1.2	BSP macro definitions	54
4.2	BSP specifications	56
4.2.1	List of abbreviations for peripherals	56
4.2.2	Naming rules	56
4.2.3	Encoding rules.....	57
4.3	BSP structure	59
4.3.1	BSP folder structure	59
4.3.2	BSP function library structure.....	60
4.3.3	Initialization and configuration for peripherals	62
4.3.4	Peripheral functions format description.....	63
5	AT32F425 peripheral library functions	64
5.1	HICK automatic clock calibration (ACC)	64
5.1.1	acc_calibration_mode_enable function.....	65
5.1.2	acc_step_set function	65
5.1.3	acc_interrupt_enable function.....	66
5.1.4	acc_hicktrim_get function.....	67

5.1.5	acc_hickcal_get function	67
5.1.6	acc_write_c1 function.....	68
5.1.7	acc_write_c2 function.....	68
5.1.8	acc_write_c3 function.....	69
5.1.9	acc_read_c1 function	69
5.1.10	acc_read_c2 function	70
5.1.11	acc_read_c3 function	70
5.1.12	acc_flag_get function	71
5.1.13	acc_interrupt_flag_get function.....	72
5.1.14	acc_flag_clear function	72
5.2	Analog-to-digital converter (ADC).....	73
5.2.1	adc_reset function.....	75
5.2.2	adc_enable function	75
5.2.3	adc_base_default_para_init function	76
5.2.4	adc_base_config function	76
5.2.5	adc_dma_mode_enable function.....	77
5.2.6	adc_interrupt_enable function.....	78
5.2.7	adc_calibration_init function.....	78
5.2.8	adc_calibration_init_status_get function.....	79
5.2.9	adc_calibration_start function	79
5.2.10	adc_calibration_status_get function.....	80
5.2.11	adc_voltage_monitor_enable function	80
5.2.12	adc_voltage_monitor_threshold_value_set function	81
5.2.13	adc_voltage_monitor_single_channel_select function	82
5.2.14	adc_ordinary_channel_set function	82
5.2.15	adc_preempt_channel_length_set function	83
5.2.16	adc_preempt_channel_set function	84
5.2.17	adc_ordinary_conversion_trigger_set function	84
5.2.18	adc_preempt_conversion_trigger_set function.....	85
5.2.19	adc_preempt_offset_value_set function	86
5.2.20	adc_ordinary_part_count_set function.....	87
5.2.21	adc_ordinary_part_mode_enable function	87
5.2.22	adc_preempt_part_mode_enable function	88
5.2.23	adc_preempt_auto_mode_enable function	88
5.2.24	adc_tempsensor_vinrv_enable function	89

5.2.25	adc_ordinary_software_trigger_enable function.....	89
5.2.26	adc_ordinary_software_trigger_status_get function.....	90
5.2.27	adc_preempt_software_trigger_enable function.....	90
5.2.28	adc_preempt_software_trigger_status_get function.....	91
5.2.29	adc_ordinary_conversion_data_get function.....	91
5.2.30	adc_preempt_conversion_data_get function.....	92
5.2.31	adc_flag_get function	92
5.2.32	adc_interrupt_flag_get function.....	93
5.2.33	adc_flag_clear 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	106
5.3.12	can_transmit_cancel function	107
5.3.13	can_message_receive function	107
5.3.14	can_receive_fifo_release function	109
5.3.15	can_receive_message_pending_get function	110
5.3.16	can_operating_mode_set function.....	110
5.3.17	can_doze_mode_enter function.....	111
5.3.18	can_doze_mode_exit function	112
5.3.19	can_error_type_record_get function	112
5.3.20	can_receive_error_counter_get function	113
5.3.21	can_transmit_error_counter_get function	113
5.3.22	can_interrupt_enable function.....	114
5.3.23	can_flag_get function	115
5.3.24	can_interrupt_flag_get function.....	116
5.3.25	can_flag_clear function	117

5.4	CRC calculation unit (CRC).....	118
5.4.1	crc_data_reset function.....	119
5.4.2	crc_one_word_calculate function.....	119
5.4.3	crc_block_calculate function	120
5.4.4	crc_data_get function.....	120
5.4.5	crc_common_data_set function	121
5.4.6	crc_common_data_get function.....	121
5.4.7	crc_init_data_set function	122
5.4.8	crc_reverse_input_data_set function.....	122
5.4.9	crc_reverse_output_data_set function.....	123
5.4.10	crc_poly_value_set function.....	123
5.4.11	crc_poly_value_get function.....	124
5.4.12	crc_poly_size_set function.....	124
5.4.13	crc_poly_size_get function.....	125
5.5	Clock and reset management (CRM)	126
5.5.1	crm_reset function.....	128
5.5.2	crm_lext_bypass function.....	128
5.5.3	crm_hext_bypass function	129
5.5.4	crm_flag_get function.....	129
5.5.5	crm_interrupt_flag_get function	130
5.5.6	crm_hext_stable_wait function.....	131
5.5.7	crm_hick_clock_trimming_set function	132
5.5.8	crm_hick_clock_calibration_set function	132
5.5.9	crm_periph_clock_enable	133
5.5.10	crm_periph_reset function.....	133
5.5.11	crm_periph_sleep_mode_clock_enable function	134
5.5.12	crm_clock_source_enable function.....	135
5.5.13	crm_flag_clear function	135
5.5.14	crm_ertc_clock_select function.....	136
5.5.15	crm_ertc_clock_enable function	137
5.5.16	crm_ahb_div_set function	137
5.5.17	crm_apb1_div_set function	138
5.5.18	crm_apb2_div_set function	138
5.5.19	crm_adc_clock_div_set function	139
5.5.20	crm_usb_clock_div_set function	139

5.5.21	crm_clock_failure_detection_enable function.....	140
5.5.22	crm_batteryPowered_domain_reset function.....	140
5.5.23	crm_pll_config function	141
5.5.24	crm_pll_config2 function	142
5.5.25	crm_sysclk_switch function.....	143
5.5.26	crm_sysclk_switch_status_get function	143
5.5.27	crm_clocks_freq_get function	144
5.5.28	crm_clock_out_set function.....	145
5.5.29	crm_interrupt_enable function	145
5.5.30	crm_auto_step_mode_enable function.....	146
5.5.31	crm_hick_sclk_frequency_select function	146
5.5.32	crm_usb_clock_source_select function	147
5.5.33	crm_clkout_div_set function.....	147
5.5.34	crm_usbddiv_reset function	148
5.5.35	crm_pll_parameter_calculate function	148
5.6	Debug.....	149
5.6.1	debug_device_id_get function	149
5.6.2	debug_periph_mode_set function.....	150
5.7	DMA controller.....	151
5.7.1	dma_default_para_init function.....	152
5.7.2	dma_init function	153
5.7.3	dma_reset function.....	155
5.7.4	dma_data_number_set function	155
5.7.5	dma_data_number_get function	156
5.7.6	dma_interrupt_enable function	156
5.7.7	dma_channel_enable function	157
5.7.8	dma_flexible_config function.....	157
5.7.9	dma_flag_get function	158
5.7.10	dma_flag_clear function	160
5.8	Real-time clock (ERTC).....	162
5.8.1	ertc_num_to_bcd function.....	164
5.8.2	ertc_bcd_to_num function.....	164
5.8.3	ertc_write_protect_enable function	165
5.8.4	ertc_write_protect_disable function	165
5.8.5	ertc_wait_update function	166

5.8.6	ertc_wait_flag function	166
5.8.7	ertc_init_mode_enter function.....	167
5.8.8	ertc_init_mode_exit function	167
5.8.9	ertc_reset function.....	168
5.8.10	ertc_divider_set function	168
5.8.11	ertc_hour_mode_set function	169
5.8.12	ertc_date_set function	169
5.8.13	ertc_time_set function	170
5.8.14	ertc_calendar_get function.....	170
5.8.15	ertc_sub_second_get function	171
5.8.16	ertc_alarm_mask_set function	172
5.8.17	ertc_alarm_week_date_select function	173
5.8.18	ertc_alarm_set function.....	174
5.8.19	ertc_alarm_sub_second_set function	175
5.8.20	ertc_alarm_enable function.....	176
5.8.21	ertc_alarm_get function.....	176
5.8.22	ertc_alarm_sub_second_get function	178
5.8.23	ertc_wakeup_clock_set function	178
5.8.24	ertc_wakeup_counter_set function	179
5.8.25	ertc_wakeup_counter_get function	179
5.8.26	ertc_wakeup_enable function	180
5.8.27	ertc_smooth_calibration_config function	180
5.8.28	ertc_cal_output_select function	181
5.8.29	ertc_cal_output_enable function	181
5.8.30	ertc_time_adjust function	182
5.8.31	ertc_daylight_set function	182
5.8.32	ertc_daylight_bpr_get function	183
5.8.33	ertc_refer_clock_detect_enable function	183
5.8.34	ertc_direct_read_enable function.....	184
5.8.35	ertc_output_set function.....	184
5.8.36	ertc_timestamp_valid_edge_set function	185
5.8.37	ertc_timestamp_enable function	185
5.8.38	ertc_timestamp_get function	186
5.8.39	ertc_timestamp_sub_second_get function	187
5.8.40	ertc_tamper_pull_up_enable function.....	187

5.8.41	ertc_tamper_preamble_set function	188
5.8.42	ertc_tamper_filter_set function.....	188
5.8.43	ertc_tamper_detect_freq_set function	189
5.8.44	ertc_tamper_valid_edge_set function.....	190
5.8.45	ertc_tamper_timestamp_enable function.....	190
5.8.46	ertc_tamper_enable function.....	191
5.8.47	ertc_interrupt_enable function	191
5.8.48	ertc_interrupt_get function	192
5.8.49	ertc_flag_get function.....	192
5.8.50	ertc_interrupt_flag_get function	193
5.8.51	ertc_flag_clear function	193
5.8.52	ertc_bpr_data_write function.....	194
5.8.53	ertc_bpr_data_read function	196
5.9	External interrupt/event controller (EXINT)	197
5.9.1	exint_reset function	198
5.9.2	exint_default_para_init function	198
5.9.3	exint_init function.....	199
5.9.4	exint_flag_clear function	200
5.9.5	exint_flag_get function	200
5.9.6	exint_interrupt_flag_get function.....	201
5.9.7	exint_software_interrupt_event_generate function.....	201
5.9.8	exint_interrupt_enable function.....	202
5.9.9	exint_event_enable function	202
5.10	Flash memory controller (FLASH)	203
5.10.1	flash_flag_get function	205
5.10.2	flash_flag_clear function	206
5.10.3	flash_operation_status_get function	206
5.10.4	flash_operation_wait_for function	207
5.10.5	flash_unlock function.....	207
5.10.6	flash_lock function.....	208
5.10.7	flash_sector_erase function	208
5.10.8	flash_internal_all_erase function	208
5.10.9	flash_user_system_data_erase function	209
5.10.10	flash_word_program function.....	209
5.10.11	flash_halfword_program function.....	210

5.10.12 flash_byte_program function.....	211
5.10.13 flash_user_system_data_program function.....	211
5.10.14 flash_epp_set function	212
5.10.15 flash_epp_status_get function	213
5.10.16 flash_fap_enable function	213
5.10.17 flash_fap_status_get function	214
5.10.18 flash_fap_high_level_enable	214
5.10.19 flash_fap_high_level_status_get.....	215
5.10.20 flash_ssb_set function.....	215
5.10.21 flash_ssb_status_get function.....	216
5.10.22 flash_interrupt_enable function.....	216
5.10.23 flash_slib_enable function.....	217
5.10.24 flash_slib_disable function	217
5.10.25 flash_slib_state_get function.....	218
5.10.26 flash_slib_start_sector_get function.....	218
5.10.27 flash_slib_inststart_sector_get function.....	219
5.10.28 flash_slib_end_sector_get function.....	219
5.10.29 flash_crc_calibrate function.....	219
5.10.30 flash_boot_memory_extension_mode_enable.....	220
5.10.31 flash_extension_memory_slib_enable.....	220
5.10.32 flash_extension_memory_slib_state_get.....	221
5.10.33 flash_em_slib_inststart_sector_get.....	221
5.10.34 flash_low_power_mode_enable	222
5.11 General-purpose I/Os and multiplexed I/Os (GPIO/IOMUX).....	223
5.11.1 gpio_reset function	224
5.11.2 gpio_init function	224
5.11.3 gpio_default_para_init function	226
5.11.4 gpio_input_data_bit_read function.....	226
5.11.5 gpio_input_data_read function.....	227
5.11.6 gpio_output_data_bit_read function.....	227
5.11.7 gpio_output_data_read function	227
5.11.8 gpio_bits_set function	228
5.11.9 gpio_bits_reset function	228
5.11.10 gpio_bits_write function.....	229
5.11.11 gpio_port_write function	229

5.11.12 gpio_pin_wp_config function.....	230
5.11.13 gpio_pins_huge_driven_config function	230
5.11.14 gpio_pin_mux_config function	231
5.12 I2C interfaces	232
5.12.1 i2c_reset function	234
5.12.2 i2c_init function.....	235
5.12.3 i2c_own_address1_set function.....	235
5.12.4 i2c_own_address2_set function.....	236
5.12.5 i2c_own_address2_enable function.....	236
5.12.6 i2c_smbus_enable function.....	237
5.12.7 i2c_enable function	238
5.12.8 i2c_clock_stretch_enable function	238
5.12.9 i2c_ack_enable function.....	239
5.12.10 i2c_addr10_mode_enable function.....	239
5.12.11 i2c_transfer_addr_set function.....	240
5.12.12 i2c_transfer_addr_get function	240
5.12.13 i2c_transfer_dir_set function	241
5.12.14 i2c_transfer_dir_get function.....	241
5.12.15 i2c_matched_addr_get function.....	242
5.12.16 i2c_auto_stop_enable function	242
5.12.17 i2c_reload_enable function	243
5.12.18 i2c_cnt_set function.....	243
5.12.19 i2c_addr10_header_enable function.....	244
5.12.20 i2c_general_call_enable function.....	244
5.12.21 i2c_smbus_alert_set function	245
5.12.22 i2c_slave_data_ctrl_enable function	245
5.12.23 i2c_pec_calculate_enable function	246
5.12.24 i2c_pec_transmit_enable function	246
5.12.25 i2c_pec_value_get function.....	247
5.12.26 i2c_timeout_set function	247
5.12.27 i2c_timeout_detct_set function	248
5.12.28 i2c_timeout_enable function	248
5.12.29 i2c_ext_timeout_set function.....	249
5.12.30 i2c_ext_timeout_enable function	249
5.12.31 i2c_interrupt_enable function	250

5.12.32 i2c_interrupt_get function.....	251
5.12.33 i2c_dma_enable function	251
5.12.34 i2c_transmit_set function	252
5.12.35 i2c_start_generate function.....	253
5.12.36 i2c_stop_generate function.....	253
5.12.37 i2c_data_send function	254
5.12.38 i2c_data_receive function	254
5.12.39 i2c_flag_get function	255
5.12.40 i2c_interrupt_flag_get function.....	256
5.12.41 i2c_flag_clear function	257
5.12.42 i2c_config function.....	258
5.12.43 i2c_lowlevel_init function.....	260
5.12.44 i2c_wait_end function.....	260
5.12.45 i2c_wait_flag function.....	261
5.12.46 i2c_master_transmit function	262
5.12.47 i2c_master_receive function	263
5.12.48 i2c_slave_transmit function.....	264
5.12.49 i2c_slave_receive function	264
5.12.50 i2c_master_transmit_int function	265
5.12.51 i2c_master_receive_int function	266
5.12.52 i2c_slave_transmit_int function.....	266
5.12.53 i2c_slave_receive_int function	267
5.12.54 i2c_master_transmit_dma function	268
5.12.55 i2c_master_receive_dma function	269
5.12.56 i2c_slave_transmit_dma function.....	269
5.12.57 i2c_slave_receive_dma function.....	270
5.12.58 i2c_smbus_master_transmit function	270
5.12.59 i2c_smbus_master_receive function	271
5.12.60 i2c_smbus_slave_transmit function	272
5.12.61 i2c_smbus_slave_receive function	273
5.12.62 i2c_memory_write function	273
5.12.63 i2c_memory_write_int function	274
5.12.64 i2c_memory_write_dma function	275
5.12.65 i2c_memory_read function	276
5.12.66 i2c_memory_read_int function	277

5.12.67 i2c_memory_read_dma function.....	278
5.12.68 i2c_evt_irq_handler function	279
5.12.69 i2c_err_irq_handler function.....	279
5.12.70 i2c_dma_tx_irq_handler function	280
5.12.71 i2c_dma_rx_irq_handler function.....	280
5.13 Nested vectored interrupt controller (NVIC).....	281
5.13.1 nvic_system_reset function.....	282
5.13.2 nvic_irq_enable function	282
5.13.3 nvic_irq_disable function.....	283
5.13.4 nvic_priority_group_config function	283
5.13.5 nvic_vector_table_set function.....	284
5.13.6 nvic_lowpower_mode_config function	285
5.14 Power controller (PWC).....	286
5.14.1 pwc_reset function	287
5.14.2 pwc_batteryPowered_domain_access function.....	287
5.14.3 pwc_pvm_level_select function	288
5.14.4 pwc_power_voltage_monitor_enable function.....	288
5.14.5 pwc_wakeup_pin_enable function	289
5.14.6 pwc_flag_clear function.....	289
5.14.7 pwc_flag_get function	290
5.14.8 pwc_sleep_mode_enter function	290
5.14.9 pwc_deep_sleep_mode_enter function	291
5.14.10 pwc_voltage_regulate_set function.....	291
5.14.11 pwc_standby_mode_enter function	292
5.15 Serial peripheral interface (SPI)/ I ² S	293
5.15.1 spi_i2s_reset function	294
5.15.2 spi_default_para_init function	294
5.15.3 spi_init function.....	295
5.15.4 spi_ti_mode_enable function	297
5.15.5 spi_crc_next_transmit function	297
5.15.6 spi_crc_polynomial_set function	298
5.15.7 spi_crc_polynomial_get function.....	298
5.15.8 spi_crc_enable function	299
5.15.9 spi_crc_value_get function.....	299
5.15.10 spi_hardware_cs_output_enable function	300

5.15.11 spi_software_cs_internal_level_set function	300
5.15.12 spi_frame_bit_num_set function	301
5.15.13 spi_half_duplex_direction_set function	301
5.15.14 spi_enable function	302
5.15.15 i2s_default_para_init function	302
5.15.16 i2s_init function.....	303
5.15.17 i2s_enable function	304
5.15.18 spi_i2s_interrupt_enable function	305
5.15.19 spi_i2s_dma_transmitter_enable function	305
5.15.20 spi_i2s_dma_receiver_enable function.....	306
5.15.21 spi_i2s_data_transmit function	306
5.15.22 spi_i2s_data_receive function.....	307
5.15.23 spi_i2s_flag_get function.....	307
5.15.24 spi_i2s_interrupt_flag_get function	308
5.15.25 spi_i2s_flag_clear function.....	309
5.16 System configuration controller (SCFG)	310
5.16.1 scfg_reset function	311
5.16.2 scfg_infrared_config function	311
5.16.3 scfg_mem_map_get function	312
5.16.4 scfg_pa11pa12_pin_remap function	312
5.16.5 scfg_exint_line_config function	313
5.16.6 scfg_pins_ultra_driven_enable function	314
5.16.7 scfg_i2s_full_duplex_config function	314
5.16.8 scfg_pvm_lock_enable function.....	315
5.17 SysTick	316
5.17.1 systick_clock_source_config function.....	316
5.17.2 SysTick_Config function.....	317
5.18 TMR	318
5.18.1 tmr_reset function.....	320
5.18.2 tmr_counter_enable function.....	320
5.18.3 tmr_output_default_para_init function.....	321
5.18.4 tmr_input_default_para_init function	321
5.18.5 tmr_brkdt_default_para_init function	322
5.18.6 tmr_base_init function	323
5.18.7 tmr_clock_source_div_set function.....	323

5.18.8 tmr_cnt_dir_set function.....	324
5.18.9 tmr_repetition_counter_set function.....	324
5.18.10 tmr_counter_value_set function.....	325
5.18.11 tmr_counter_value_get function.....	325
5.18.12 tmr_div_value_set function	326
5.18.13 tmr_div_value_get function	326
5.18.14 tmr_output_channel_config function	327
5.18.15 tmr_output_channel_mode_select function	329
5.18.16 tmr_period_value_set function	330
5.18.17 tmr_period_value_get function.....	330
5.18.18 tmr_channel_value_set function	331
5.18.19 tmr_channel_value_get function	332
5.18.20 tmr_period_buffer_enable function	332
5.18.21 tmr_output_channel_buffer_enable function	333
5.18.22 tmr_output_channel_immediately_set function	334
5.18.23 tmr_output_channel_switch_set function.....	335
5.18.24 tmr_one_cycle_mode_enable function	335
5.18.25 tmr_32_bit_function_enable function	336
5.18.26 tmr_overflow_request_source_set function	336
5.18.27 tmr_overflow_event_disable function.....	337
5.18.28 tmr_input_channel_init function	337
5.18.29 tmr_channel_enable function	339
5.18.30 tmr_input_channel_filter_set function	340
5.18.31 tmr_pwm_input_config function	340
5.18.32 tmr_channel1_input_select function	341
5.18.33 tmr_input_channel_divider_set function	342
5.18.34 tmr_primary_mode_select function.....	343
5.18.35 tmr_sub_mode_select function	344
5.18.36 tmr_channel_dma_select function	345
5.18.37 tmr_hall_select function	345
5.18.38 tmr_channel_buffer_enable function.....	345
5.18.39 tmr_trigger_input_select function.....	347
5.18.40 tmr_sub_sync_mode_set function	347
5.18.41 tmr_dma_request_enable function	348
5.18.42 tmr_interrupt_enable function	348

5.18.43 tmr_interrupt_flag_get function	349
5.18.44 tmr_flag_get function.....	350
5.18.45 tmr_flag_clear function.....	350
5.18.46 tmr_event_sw_trigger function.....	351
5.18.47 tmr_output_enable function.....	351
5.18.48 tmr_internal_clock_set function	352
5.18.49 tmr_output_channel_polarity_set function	353
5.18.50 tmr_external_clock_config function.....	354
5.18.51 tmr_external_clock_mode1_config function	355
5.18.52 tmr_external_clock_mode2_config function	356
5.18.53 tmr_encoder_mode_config function.....	357
5.18.54 tmr_force_output_set function	358
5.18.55 tmr_dma_control_config function.....	359
5.18.56 tmr_brkdt_config function.....	360
5.18.57 tmr_brk_filter_value_get function.....	362
5.18.58 tmr_iremap_config function.....	363
5.19 Universal synchronous/asynchronous receiver/transmitter (USART)	364
5.19.1 usart_reset function.....	365
5.19.2 usart_init function	366
5.19.3 usart_parity_selection_config function.....	367
5.19.4 usart_enable function.....	367
5.19.5 usart_transmitter_enable function.....	368
5.19.6 usart_receiver_enable function.....	368
5.19.7 usart_clock_config function.....	369
5.19.8 usart_clock_enable function.....	370
5.19.9 usart_interrupt_enable function	370
5.19.10 usart_dma_transmitter_enable function.....	371
5.19.11 usart_dma_receiver_enable function.....	371
5.19.12 usart_wakeup_id_set function	372
5.19.13 usart_wakeup_mode_set function	372
5.19.14 usart_receiver_mute_enable function.....	373
5.19.15 usart_break_bit_num_set function	373
5.19.16 usart_lin_mode_enable function	374
5.19.17 usart_data_transmit function.....	374
5.19.18 usart_data_receive function	375

5.19.19 usart_break_send function	375
5.19.20 usart_smartcard_guard_time_set function	376
5.19.21 usart_irda_smartcard_division_set function	376
5.19.22 usart_smartcard_mode_enable function	377
5.19.23 usart_smartcard_nack_set function	377
5.19.24 usart_single_line_halfduplex_select function	378
5.19.25 usart_irda_mode_enable function	378
5.19.26 usart_irda_low_power_enable function	379
5.19.27 usart_hardware_flow_control_set function	379
5.19.28 usart_flag_get function	380
5.19.29 usart_interrupt_flag_get function	381
5.19.30 usart_flag_clear function	382
5.19.31 usart_rs485_delay_time_config function	382
5.19.32 usart_transmit_receive_pin_swap function	383
5.19.33 usart_id_bit_num_set function	383
5.19.34 usart_de_polarity_set function	384
5.19.35 usart_rs485_mode_enable function	384
5.20 Watchdog timer (WDT)	385
5.20.1 wdt_enable function	386
5.20.2 wdt_counter_reload function	386
5.20.3 wdt_reload_value_set function	387
5.20.4 wdt_divider_set function	387
5.20.5 wdt_register_write_enable function	388
5.20.6 wdt_flag_get function	388
5.20.7 wdt_window_counter_set function	389
5.21 Window watchdog timer (WWDT)	390
5.21.1 wwdt_reset function	391
5.21.2 wwdt_divider_set function	391
5.21.3 wwdt_enable function	392
5.21.4 wwdt_interrupt_enable function	392
5.21.5 wwdt_counter_set function	392
5.21.6 wwdt_window_counter_set function	393
5.21.7 wwdt_flag_get function	393
5.21.8 wwdt_interrupt_flag_get function	394
5.21.9 wwdt_flag_clear function	394

6	Precautions	395
6.1	Device model replacement	395
6.1.1	KEIL environment.....	395
6.1.2	IAR environment.....	396
6.2	Unable to identify IC by JLink software in Keil	398
6.3	How to change HEXT crystal	400
7	Revision history.....	401

List of tables

Table 1. Summary of macro definitions	54
Table 2. List of abbreviations for peripherals.....	56
Table 3. Summary of BSP function library files	62
Table 4. Function format description for peripherals	63
Table 5. Summary of ACC registers	64
Table 6. Summary of ACC library functions.....	64
Table 7. acc_calibration_mode_enable function	65
Table 8. acc_step_set function	65
Table 9. acc_interrupt_enable function.....	66
Table 10. acc_hicktrim_get function	67
Table 11. acc_hickcal_get function	67
Table 12. acc_write_c1 function	68
Table 13. acc_write_c2 function	68
Table 14. acc_write_c3 function	69
Table 15. acc_read_c1 function.....	69
Table 16. acc_read_c2 function.....	70
Table 17. acc_read_c3 function.....	70
Table 18. acc_flag_get function	71
Table 19. acc_flag_clear function	72
Table 20. acc_flag_clear function	72
Table 21. Summary of ADC registers	73
Table 22. Summary of ADC library functions.....	74
Table 23. adc_reset function.....	75
Table 24. adc_enable function.....	75
Table 25. adc_base_default_para_init function	76
Table 26. adc_base_config function	76
Table 27. adc_dma_mode_enable function.....	77
Table 28. adc_interrupt_enable function	78
Table 29. adc_calibration_init function.....	78
Table 30. adc_calibration_init_status_get function.....	79
Table 31. adc_calibration_start function	79
Table 32. adc_calibration_status_get function	80
Table 33. adc_voltage_monitor_enable function.....	80
Table 34. adc_voltage_monitor_threshold_value_set function	81

Table 35. adc_voltage_monitor_single_channel_select function	82
Table 36. adc_ordinary_channel_set function	82
Table 37. adc_preempt_channel_length_set function	83
Table 38. adc_preempt_channel_set function	84
Table 39. adc_ordinary_conversion_trigger_set function	84
Table 40. adc_preempt_conversion_trigger_set function	85
Table 41. adc_preempt_offset_value_set function	86
Table 42. adc_ordinary_part_count_set function	87
Table 43. adc_ordinary_part_mode_enable function	87
Table 44. adc_preempt_part_mode_enable function	88
Table 45. adc_preempt_auto_mode_enable function	88
Table 46. adc_tempersensor_vinrv_enable function	89
Table 47. adc_ordinary_software_trigger_enable function	89
Table 48. adc_ordinary_software_trigger_status_get function	90
Table 49. adc_preempt_software_trigger_enable function	90
Table 50. adc_preempt_software_trigger_status_get function	91
Table 51. adc_ordinary_conversion_data_get function	91
Table 52. adc_preempt_conversion_data_get function	92
Table 53. adc_flag_get function	92
Table 54. adc_interrupt_flag_get function	93
Table 55. adc_flag_clear function	94
Table 56. Summary of CAN registers	95
Table 57. Summary of CAN library functions	96
Table 58. can_reset function	97
Table 59. can_baudrate_default_para_init function	97
Table 60. can_baudrate_set function	98
Table 61. can_default_para_init function	99
Table 62. can_base_init function	99
Table 63. can_filter_default_para_init function	101
Table 64. can_filter_init function	101
Table 65. can_debug_transmission_prohibit function	103
Table 66. can_ttc_mode_enable function	103
Table 67. can_message_transmit function	104
Table 68. can_transmit_status_get function	106
Table 69. can_transmit_cancel function	107

Table 70. can_message_receive function	107
Table 71. can_receive_fifo_release function	109
Table 72. can_receive_message_pending_get function	110
Table 73. can_operating_mode_set function.....	110
Table 74. can_doze_mode_enter function	111
Table 75. can_doze_mode_exit function	112
Table 76. can_error_type_record_get function.....	112
Table 77. can_receive_error_counter_get function	113
Table 78. can_transmit_error_counter_get function.....	113
Table 79. can_interrupt_enable function	114
Table 80. can_flag_get function.....	115
Table 81. can_interrupt_flag_get function	116
Table 82. can_flag_clear function	117
Table 83. Summary of CRC registers	118
Table 84. Summary of CRC library functions	118
Table 85. crc_data_reset function.....	119
Table 86. crc_one_word_calculate function	119
Table 87. crc_block_calculate function	120
Table 88. crc_data_get function.....	120
Table 89. crc_common_data_set function.....	121
Table 90. crc_common_data_get function.....	121
Table 91. crc_init_data_set function	122
Table 92. crc_reverse_input_data_set function.....	122
Table 93. crc_reverse_output_data_set function.....	123
Table 94. crc_poly_value_set function.....	123
Table 95. crc_poly_value_get function	124
Table 96. crc_poly_size_set function.....	124
Table 97. crc_poly_size_get function.....	125
Table 98. Summary of CRM registers.....	126
Table 99. Summary of CRM library functions	127
Table 100. crm_reset function.....	128
Table 101. crm_lext_bypass function	128
Table 102. crm_hext_bypass function	129
Table 103. crm_flag_get function.....	129
Table 104. crm_interrupt_flag_get function	130

Table 105. crm_hext_stable_wait function	131
Table 106. crm_hick_clock_trimming_set function.....	132
Table 107. crm_hick_clock_calibration_set function	132
Table 108. crm_periph_clock_enable function	133
Table 109. crm_periph_reset function	133
Table 110. crm_periph_sleep_mode_clock_enable function.....	134
Table 111. crm_clock_source_enable function	135
Table 112. crm_flag_clear function	135
Table 113. crm_ertc_clock_select function.....	136
Table 114. crm_ertc_clock_enable function.....	137
Table 115. crm_ahb_div_set function	137
Table 116. crm_apb1_div_set function	138
Table 117. crm_apb2_div_set function	138
Table 118. crm_adc_clock_div_set function	139
Table 119. crm_usb_clock_div_set function	139
Table 120. crm_clock_failure_detection_enable function.....	140
Table 121. crm_batteryPowered_domain_reset	140
Table 122. crm_pll_config function	141
Table 123. crm_pll_config2 function	142
Table 124. crm_sysclk_switch function.....	143
Table 125. crm_sysclk_switch_status_get function.....	143
Table 126. crm_clocks_freq_get function	144
Table 127. crm_clock_out_set function	145
Table 128. crm_interrupt_enable function	145
Table 129. crm_auto_step_mode_enable function	146
Table 130. crm_hick_sclk_frequency_select function	146
Table 131. crm_usb_clock_source_select function	147
Table 132. crm_clkout_div_set function.....	147
Table 133. crm_usbdv_reset function	148
Table 134. crm_pll_parameter_calculate function	148
Table 135. Summary of DEBUG registers	149
Table 136. Summary of DEBUG library functions	149
Table 137. debug_device_id_get function	149
Table 138. debug_periph_mode_set function	150
Table 139. Summary of DMA registers	151

Table 140. Summary of DMA library functions	152
Table 141. dma_default_para_init function.....	152
Table 142. dma_init_struct default values	153
Table 143. dma_init function	153
Table 144. dma_reset function	155
Table 145. dma_data_number_set function	155
Table 146. dma_data_number_get function	156
Table 147. dma_interrupt_enable function	156
Table 148. dma_channel_enable function.....	157
Table 149. dma_flexible_config function	157
Table 150. Flexible mapping request source ID	158
Table 151. dma_flag_get function.....	158
Table 152. dma_flag_clear function.....	160
Table 153. Summary of ERTC registers	162
Table 154. Summary of ERTC library functions	162
Table 155. ertc_num_to_bcd function.....	164
Table 156. ertc_bcd_to_num function.....	164
Table 157. ertc_write_protect_enable function.....	165
Table 158. ertc_write_protect_disable function	165
Table 159. ertc_wait_update function.....	166
Table 160. ertc_wait_flag function	166
Table 161. ertc_init_mode_enter function	167
Table 162. ertc_init_mode_exit function	167
Table 163. ertc_reset function.....	168
Table 164. ertc_divider_set function	168
Table 165. ertc_hour_mode_set function	169
Table 166. ertc_date_set function.....	169
Table 167. ertc_time_set function.....	170
Table 168. ertc_calendar_get function.....	170
Table 169. ertc_sub_second_get function	171
Table 170. ertc_alarm_mask_set function	172
Table 171. ertc_alarm_week_date_select function	173
Table 172. ertc_alarm_set function.....	174
Table 173. ertc_alarm_sub_second_set function	175
Table 174. ertc_alarm_enable function.....	176

Table 175. ertc_alarm_get function	176
Table 176. ertc_alarm_sub_second_get function.....	178
Table 177. ertc_wakeup_clock_set function.....	178
Table 178. ertc_wakeup_counter_set function	179
Table 179. ertc_wakeup_counter_get function.....	179
Table 180. ertc_wakeup_enable function	180
Table 181. ertc_smooth_calibration_config function	180
Table 182. ertc_cal_output_select function	181
Table 183. ertc_cal_output_enable function.....	181
Table 184. ertc_time_adjust function	182
Table 185. ertc_daylight_set function	182
Table 186. ertc_daylight_bpr_get function.....	183
Table 187. ertc_refer_clock_detect_enable function.....	183
Table 188. ertc_direct_read_enable function	184
Table 189. ertc_output_set function.....	184
Table 190. ertc_timestamp_valid_edge_set function	185
Table 191. ertc_timestamp_enable function.....	185
Table 192. ertc_timestamp_get function.....	186
Table 193. ertc_timestamp_sub_second_get function	187
Table 194. ertc_tamper_pull_up_enable function.....	187
Table 195. ertc_tamper_purge_set function	188
Table 196. ertc_tamper_filter_set function	188
Table 197. ertc_tamper_detect_freq_set function	189
Table 198. ertc_tamper_valid_edge_set function.....	190
Table 199. ertc_tamper_timestamp_enable function	190
Table 200. ertc_tamper_enable function	191
Table 201. ertc_interrupt_enable function	191
Table 202. ertc_interrupt_get function	192
Table 203. ertc_flag_get function.....	192
Table 204. ertc_interrupt_flag_get function	193
Table 205. ertc_flag_clear function.....	193
Table 206. ertc_bpr_data_write function	194
Table 207. ertc_bpr_data_read function.....	196
Table 208. Summary of EXINT registers	197
Table 209. Summary of EXINT library functions.....	197

Table 210. exint_reset function	198
Table 211. exint_default_para_init function	198
Table 212. exint_init function	199
Table 213. exint_flag_clear function	200
Table 214. exint_flag_get function	200
Table 215. exint_interrupt_flag_get function.....	201
Table 216. exint_software_interrupt_event_generate function	201
Table 217. exint_interrupt_enable function.....	202
Table 218. exint_event_enable function	202
Table 219. Summary of FLASH registers	203
Table 220. Summary of FLASH library functions.....	204
Table 221. flash_flag_get function.....	205
Table 222. flash_flag_clear function	206
Table 223. flash_operation_status_get function.....	206
Table 224. flash_operation_wait_for function	207
Table 225. flash_unlock function	207
Table 226. flash_lock function.....	208
Table 227. flash_sector_erase function	208
Table 228. flash_internal_all_erase function	208
Table 229. flash_user_system_data_erase function	209
Table 230. flash_word_program function	209
Table 231. flash_halfword_program function.....	210
Table 232. flash_byte_program function.....	211
Table 233. flash_user_system_data_program function.....	211
Table 234. flash_epp_set function	212
Table 235. flash_epp_status_get function	213
Table 236. flash_fap_enable function	213
Table 237. flash_fap_status_get function	214
Table 238. flash_fap_high_level_enable function.....	214
Table 239. flash_fap_high_level_status_get function.....	215
Table 240. flash(ssb)_set function	215
Table 241. flash(ssb)_status_get function	216
Table 242. flash_interrupt_enable function.....	216
Table 243. flash_slib_enable function.....	217
Table 244. flash_slib_disable function	217

Table 245. flash_slib_state_get function	218
Table 246. flash_slib_start_sector_get function	218
Table 247. flash_slib_inststart_sector_get function.....	219
Table 248. flash_slib_end_sector_get function	219
Table 249. flash_crc_calibrate function	219
Table 250. flash_boot_memory_extension_mode_enable.....	220
Table 251. flash_extension_memory_slib_enable	220
Table 252. flash_extension_memory_slib_state_get	221
Table 253. flash_em_slib_inststart_sector_get	221
Table 254. flash_low_power_mode_enable	222
Table 255. Summary of GPIO registers.....	223
Table 256. GPIO and IOMUX library functions.....	223
Table 257. gpio_reset function.....	224
Table 258. gpio_init function	224
Table 259. gpio_default_para_init function.....	226
Table 260. gpio_init_struct default values	226
Table 261. gpio_input_data_bit_read function.....	226
Table 262. gpio_input_data_read function	227
Table 263. gpio_output_data_bit_read function	227
Table 264. gpio_output_data_read function	227
Table 265. gpio_bits_set function	228
Table 266. gpio_bits_reset function	228
Table 267. gpio_bits_write function	229
Table 268. gpio_port_write function	229
Table 269. gpio_pin_wp_config function	230
Table 270. gpio_pins_huge_driven_config function	230
Table 271. gpio_pin_mux_config function	231
Table 272. Summary of I ² C register.....	232
Table 273. Summary of I ² C library functions	232
Table 274. I ² C application-layer library functions	233
Table 275. i2c_reset function	234
Table 276. i2c_init function	235
Table 277. i2c_own_address1_set function	235
Table 278. i2c_own_address2_set function	236
Table 279. i2c_own_address2_enable function	236

Table 280. i2c_smbus_enable function	237
Table 281. i2c_enable function	238
Table 282. i2c_clock_stretch_enable function.....	238
Table 283. i2c_ack_enable function	239
Table 284. i2c_addr10_mode_enable function.....	239
Table 285. i2c_transfer_addr_set function	240
Table 286. i2c_transfer_addr_get function	240
Table 287. i2c_transfer_dir_set function.....	241
Table 288. i2c_transfer_dir_get function.....	241
Table 289. i2c_matched_addr_get function.....	242
Table 290. i2c_auto_stop_enable function	242
Table 291. i2c_reload_enable function.....	243
Table 292. i2c_cnt_set function	243
Table 293. i2c_addr10_header_enable function	244
Table 294. i2c_general_call_enable function	244
Table 295. i2c_smbus_alert_set function	245
Table 296. i2c_start_generate function.....	245
Table 297. i2c_pec_calculate_enable.....	246
Table 298. i2c_pec_transmit_enable function	246
Table 299. i2c_pec_value_get function	247
Table 300. i2c_timeout_set function	247
Table 301. i2c_timeout_detct_set function	248
Table 302. i2c_timeout_enable function	248
Table 303. i2c_ext_timeout_set function	249
Table 304. i2c_ext_timeout_enable function	249
Table 305. i2c_interrupt_enable function.....	250
Table 306. i2c_interrupt_get function.....	251
Table 307. i2c_dma_enable function	251
Table 308. i2c_transmit_set function	252
Table 309. i2c_slave_transmit function.....	253
Table 310. i2c_stop_generate function.....	253
Table 311. i2c_data_send function	254
Table 312. i2c_data_receive function	254
Table 313. i2c_flag_get function	255
Table 314. i2c_interrupt_flag_get function.....	256

Table 315. i2c_flag_clear function	257
Table 316. i2c_config function	258
Table 317. i2c_lowlevel_init function	260
Table 318. i2c_wait_end function	260
Table 319. i2c_wait_flag function.....	261
Table 320. i2c_master_transmit function.....	262
Table 321. i2c_master_receivefunction	263
Table 322. i2c_slave_transmit function.....	264
Table 323. i2c_slave_receive function.....	264
Table 324. i2c_master_transmit_int function	265
Table 325. i2c_master_receive_int function	266
Table 326. i2c_master_receive_int function	266
Table 327. i2c_master_receive_int function	267
Table 328. i2c_master_transmit_dma function.....	268
Table 329. i2c_master_receive_dma function	269
Table 330. i2c_slave_transmit_dma function	269
Table 331. i2c_slave_transmit_dma function	270
Table 332. i2c_smbus_master_transmit function	270
Table 333. i2c_smbus_master_receive function	271
Table 334. i2c_smbus_slave_transmit function.....	272
Table 335. i2c_smbus_slave_receive function	273
Table 336. i2c_memory_write function	273
Table 337. i2c_memory_write_int function	274
Table 338. i2c_memory_write_dma function	275
Table 339. i2c_memory_write_dma function	276
Table 340. i2c_memory_write_dma function	277
Table 341. i2c_memory_write_dma function	278
Table 342. i2c_evt_irq_handler function.....	279
Table 343. i2c_err_irq_handler function	279
Table 344. i2c_dma_tx_irq_handler function.....	280
Table 345. i2c_dma_rx_irq_handler function.....	280
Table 346. Summary of NVIC registers	281
Table 347. Summary of NVIC library functions.....	281
Table 348. nvic_system_reset function.....	282
Table 349. nvic_irq_enable function	282

Table 350. nvic_irq_disable function.....	283
Table 351. nvic_priority_group_config function	283
Table 352. nvic_vector_table_set function	284
Table 353. nvic_lowpower_mode_config function.....	285
Table 354. Summary of PWC registers	286
Table 355. Summary of PWC library functions.....	286
Table 356. pwc_reset function	287
Table 357. pwc_batteryPowered_domain_access function.....	287
Table 358. pwc_pvm_level_select function	288
Table 359. pwc_power_voltage_monitor_enable function	288
Table 360. pwc_wakeup_pin_enable function.....	289
Table 361. pwc_flag_clear function	289
Table 362. pwc_flag_get function	290
Table 363. pwc_sleep_mode_enter function	290
Table 364. pwc_deep_sleep_mode_enter function.....	291
Table 365. pwc_voltage_regulate_set function	291
Table 366. pwc_standby_mode_enter function	292
Table 367. Summary of SPI registers	293
Table 368. Summary of SPI library functions	293
Table 369. spi_i2s_reset function	294
Table 370. spi_default_para_init function	294
Table 371. spi_init function	295
Table 372. spi_ti_mode_enable function	297
Table 373. spi_crc_next_transmit function	297
Table 374. spi_crc_polynomial_set function.....	298
Table 375. spi_crc_polynomial_get function.....	298
Table 376. spi_crc_enable function	299
Table 377. spi_crc_value_get function	299
Table 378. spi_hardware_cs_output_enable function	300
Table 379. spi_software_cs_internal_level_set function	300
Table 380. spi_frame_bit_num_set function	301
Table 381. spi_half_duplex_direction_set function.....	301
Table 382. spi_enable function	302
Table 383. i2s_default_para_init function	302
Table 384. i2s_init function	303

Table 385. i2s_enable function	304
Table 386. spi_i2s_interrupt_enable function	305
Table 387. spi_i2s_dma_transmitter_enable function	305
Table 388. spi_i2s_dma_receiver_enable function	306
Table 389. spi_i2s_data_transmit function	306
Table 390. spi_i2s_data_receive function	307
Table 391. spi_i2s_flag_get function	307
Table 392. spi_i2s_interrupt_flag_get function	308
Table 393. spi_i2s_flag_clear function.....	309
Table 394. Summary of SCFG registers.....	310
Table 395. Summary of SCFG library functions	310
Table 396. scfg_reset function.....	311
Table 397. scfg_infrared_config function.....	311
Table 398. scfg_mem_map_get function.....	312
Table 399. scfg_pa11pa12_pin_remap function.....	312
Table 400. scfg_exint_line_config function.....	313
Table 401. scfg_pins_ultra_driven_enable function	314
Table 402. scfg_i2s_full_duplex_config function	314
Table 403. scfg_adc_dma_channel_remap function.....	315
Table 404. Summary of SysTick registers	316
Table 405. Summary of SysTick library functions.....	316
Table 406. systick_clock_source_config function.....	316
Table 407. SysTick_Config function.....	317
Table 408. Summary of TMR registers	318
Table 409. Summary of TMR library functions	319
Table 410. tmr_reset function	320
Table 411. tmr_counter_enable function.....	320
Table 412. tmr_output_default_para_init function	321
Table 413. tmr_output_struct default values.....	321
Table 414. tmr_input_default_para_init function.....	321
Table 415. tmr_input_struct default values	322
Table 416. tmr_brkdt_default_para_init function	322
Table 417. tmr_brkdt_struct default values.....	322
Table 418. tmr_base_init function	323
Table 419. tmr_clock_source_div_set function.....	323

Table 420. tmr_cnt_dir_set function.....	324
Table 421. tmr_repetition_counter_set function	324
Table 422. tmr_counter_value_set function.....	325
Table 423. tmr_counter_value_get function	325
Table 424. tmr_div_value_set function	326
Table 425. tmr_div_value_get function	326
Table 426. tmr_output_channel_config function.....	327
Table 427. tmr_output_channel_mode_select function.....	329
Table 428. tmr_period_value_set function.....	330
Table 429. tmr_period_value_get function	330
Table 430. tmr_channel_value_set function	331
Table 431. tmr_channel_value_get function	332
Table 432. tmr_period_buffer_enable function	332
Table 433. tmr_output_channel_buffer_enable function	333
Table 434. tmr_output_channel_immediately_set function	334
Table 435. tmr_output_channel_switch_set function	335
Table 436. tmr_one_cycle_mode_enable function.....	335
Table 437. tmr_32_bit_function_enable function.....	336
Table 438. tmr_overflow_request_source_set function.....	336
Table 439. tmr_overflow_event_disable function	337
Table 440. tmr_input_channel_init function	337
Table 441. tmr_channel_enable function.....	339
Table 442. tmr_input_channel_filter_set function	340
Table 443. tmr_pwm_input_config function	340
Table 444. tmr_channel1_input_select function	341
Table 445. tmr_input_channel_divider_set function	342
Table 446. tmr_primary_mode_select function.....	343
Table 447. tmr_sub_mode_select function.....	344
Table 448. tmr_channel_dma_select function	345
Table 449. tmr_hall_select function	345
Table 450. tmr_channel_buffer_enable function	346
Table 451. tmr_trigger_input_select function.....	347
Table 452. tmr_sub_sync_mode_set function	347
Table 453. tmr_dma_request_enable function	348
Table 454. tmr_interrupt_enable function	348

Table 455. tmr_interrupt_flag_get function	349
Table 456. tmr_flag_get function	350
Table 457. tmr_flag_clear function.....	350
Table 458. tmr_event_sw_trigger function.....	351
Table 459. tmr_output_enable function	351
Table 460. tmr_internal_clock_set function	352
Table 461. tmr_output_channel_polarity_set function.....	353
Table 462. tmr_external_clock_config function	354
Table 463. tmr_external_clock_mode1_config function	355
Table 464. tmr_external_clock_mode2_config function	356
Table 465. tmr_encoder_mode_config function	357
Table 466. tmr_force_output_set function	358
Table 467. tmr_dma_control_config function.....	359
Table 468. tmr_brkdt_config function.....	360
Table 469. tmr_brk_filter_value_set function.....	362
Table 470. tmr_iremap_config function.....	363
Table 471. Summary of USART registers.....	364
Table 472. Summary of USART library functions	364
Table 473. usart_reset function	365
Table 474. usart_init function.....	366
Table 475. usart_parity_selection_config function	367
Table 476. usart_enable function.....	367
Table 477. usart_transmitter_enable function	368
Table 478. usart_receiver_enable function.....	368
Table 479. usart_clock_config function.....	369
Table 480. usart_clock_enable function	370
Table 481. usart_interrupt_enable function	370
Table 482. usart_dma_transmitter_enable function	371
Table 483. usart_dma_receiver_enable function.....	371
Table 484. usart_wakeup_id_set function	372
Table 485. usart_wakeup_mode_set function	372
Table 486. usart_receiver_mute_enable function.....	373
Table 487. usart_break_bit_num_set function.....	373
Table 488. usart_lin_mode_enable function.....	374
Table 489. usart_data_transmit function.....	374

Table 490. usart_data_receive function.....	375
Table 491. usart_break_send function.....	375
Table 492. usart_smartcard_guard_time_set function	376
Table 493. usart_irda_smartcard_division_set function	376
Table 494. usart_smartcard_mode_enable function	377
Table 495. usart_smartcard_nack_set function.....	377
Table 496. usart_single_line_halfduplex_select function	378
Table 497. usart_irda_mode_enable function	378
Table 498. usart_irda_low_power_enable function	379
Table 499. usart_hardware_flow_control_set function	379
Table 500. usart_flag_get function.....	380
Table 501. usart_interrupt_flag_get function	381
Table 502. usart_flag_clear function.....	382
Table 503. usart_flag_clear function.....	382
Table 504. usart_transmit_receive_pin_swap function	383
Table 505. usart_transmit_receive_pin_swap function	383
Table 506. usart_de_polarity_set function.....	384
Table 507. usart_rs485_mode_enable function	384
Table 508. Summary of WDT registers.....	385
Table 509. Summary of WDT library functions	385
Table 510. wdt_enable function	386
Table 511. wdt_counter_reload function.....	386
Table 512. wdt_reload_value_set function	387
Table 513. wdt_divider_set function	387
Table 514. wdt_register_write_enable function	388
Table 515. wdt_flag_get function.....	388
Table 516. wdt_window_counter_set function.....	389
Table 517. Summary of WWDT registers	390
Table 518. Summary of WWDT library functions.....	390
Table 519. wwdt_reset function	391
Table 520. wwdt_divider_set function.....	391
Table 521. wwdt_enable function	392
Table 522. wwdt_interrupt_enable function	392
Table 523. wwdt_counter_set function	392
Table 524. wwdt_window_counter_set function	393

Table 525. wwdt_flag_get function	393
Table 526. wwdt_interrupt_flag_get function	394
Table 527. wwdt_flag_clear function.....	394
Table 528. Clock configuration guideline	400
Table 529. Document revision history.....	401

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.....	41
Figure 7. Keil_v4 Pack installation process	42
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.....	45
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	54
Figure 26. Peripheral enable macro definitions.....	55
Figure 27. BSP folder structure	60
Figure 28. BSP function library structure	60
Figure 29. Change device part number in Keil	395
Figure 30. Change macro definition in Keil	396
Figure 31. Change device part number in IAR	397
Figure 32. Change macro definition in IAR	398
Figure 33. Error warning 1	398
Figure 34. Error warning 2	398

Figure 35. Error warning 3	399
Figure 36. JLinkLog and JLinkSettings.....	399
Figure 37. Unspecified Cortex-M4	399
Figure 38. AT32_New_Clock_Configuration window	400

1 Overview

In order to help users 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

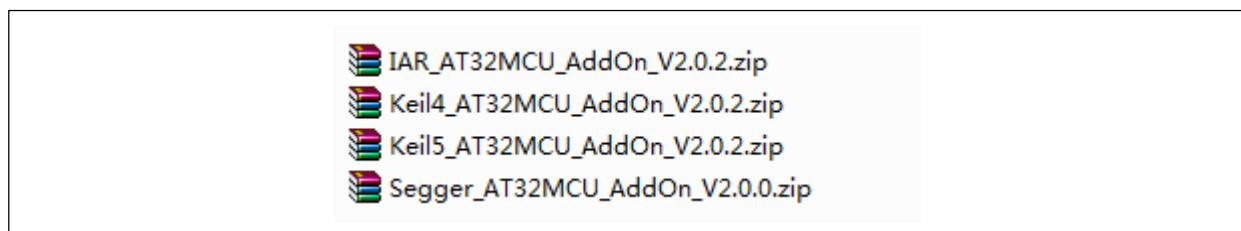
Artery Pack supports various development environment such as Keil_v5, Keil_v4, IAR_v6, IAR_v7 and IAR_v8.

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

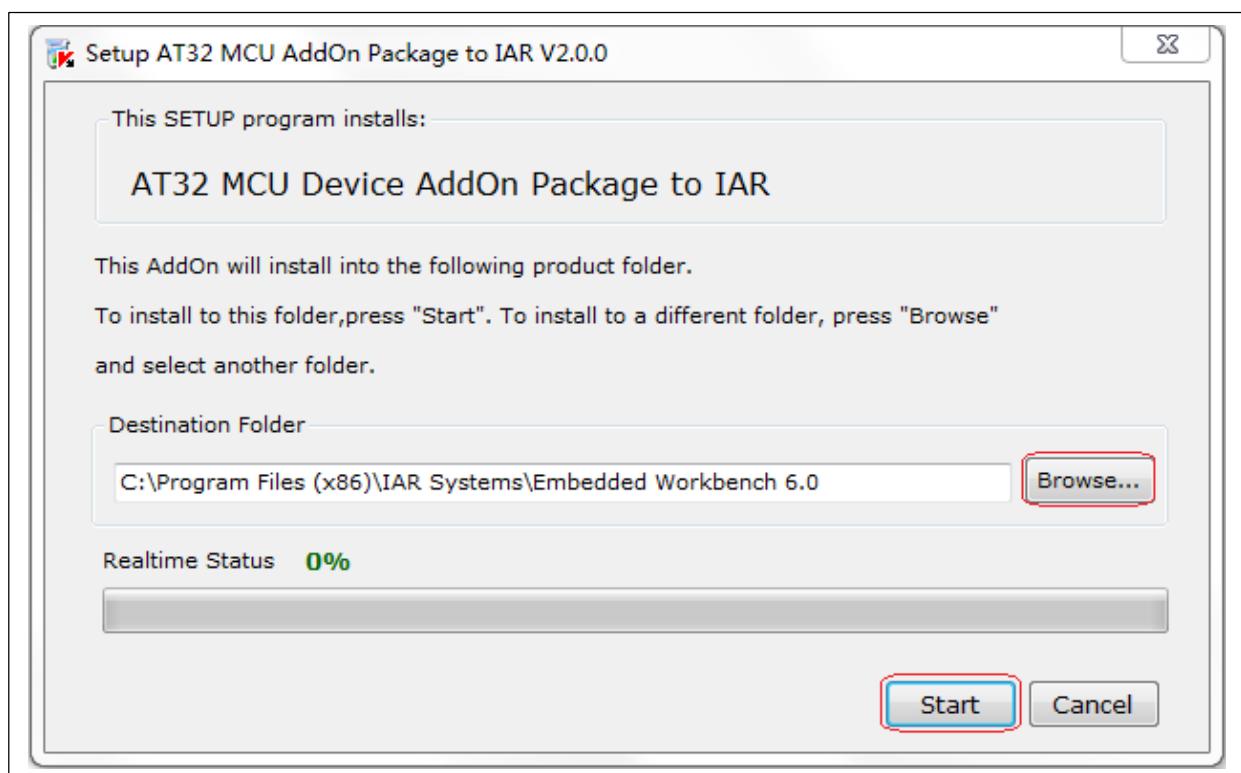


2.1 IAR Pack installation

IAR_AT32MCU_AddOn.zip: This is a zip file supporting IAR_V6, IAR_V7 and IAR_V8. Follow the steps below to install:

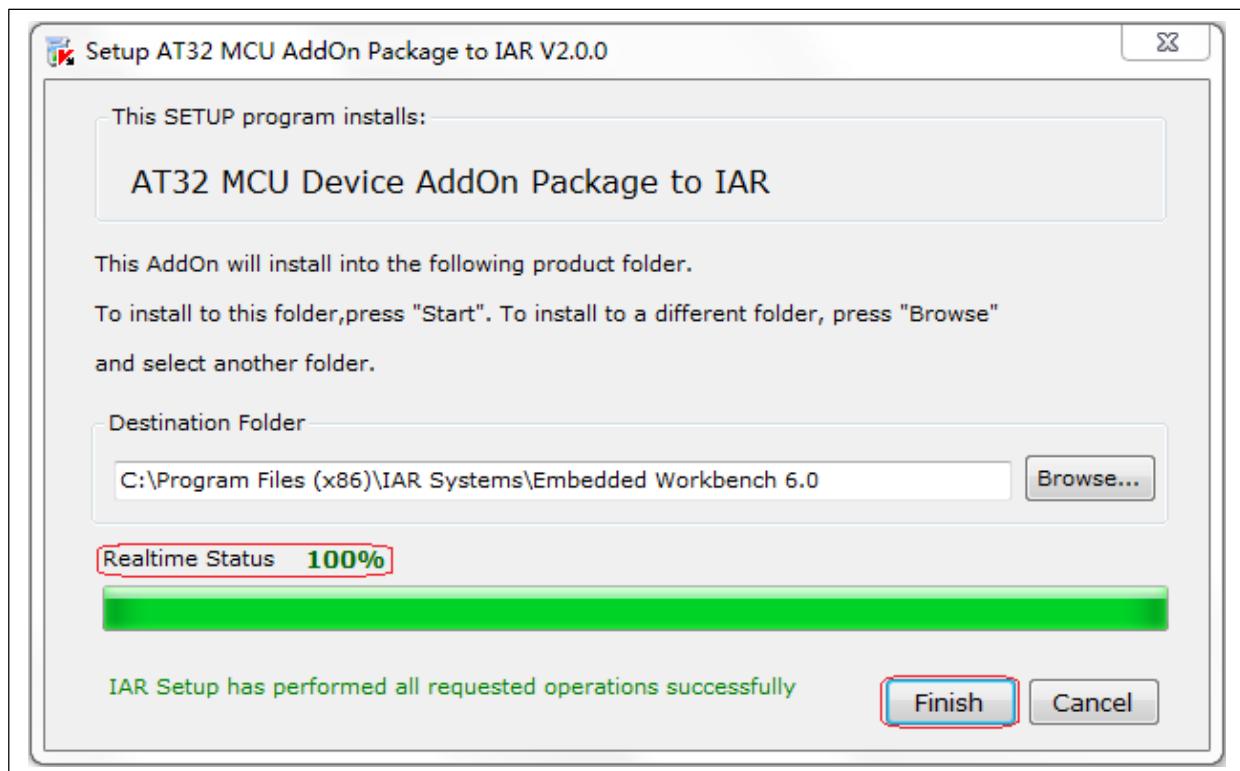
- ① Unzip *IAR_AT32MCU_AddOn.zip*;
- ② Double click on *IAR_AT32MCU_AddOn.exe*, and a dialog box pops up below (the specific version information is subject to the actual conditions).

Figure 2. IAR Pack installation window



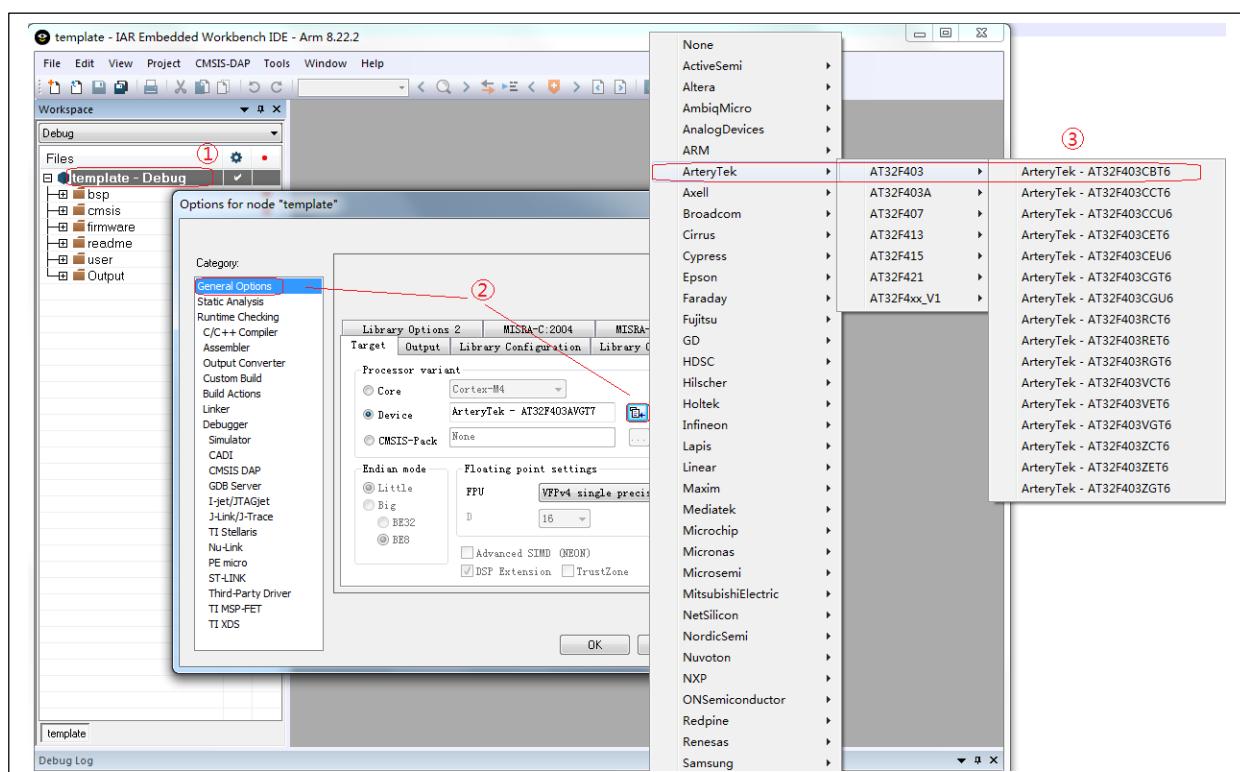
Note: If the installation path of IAR does not match the Destination Folder, click on "Browse" to select a correct path, then click on "Start", as shown below.

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

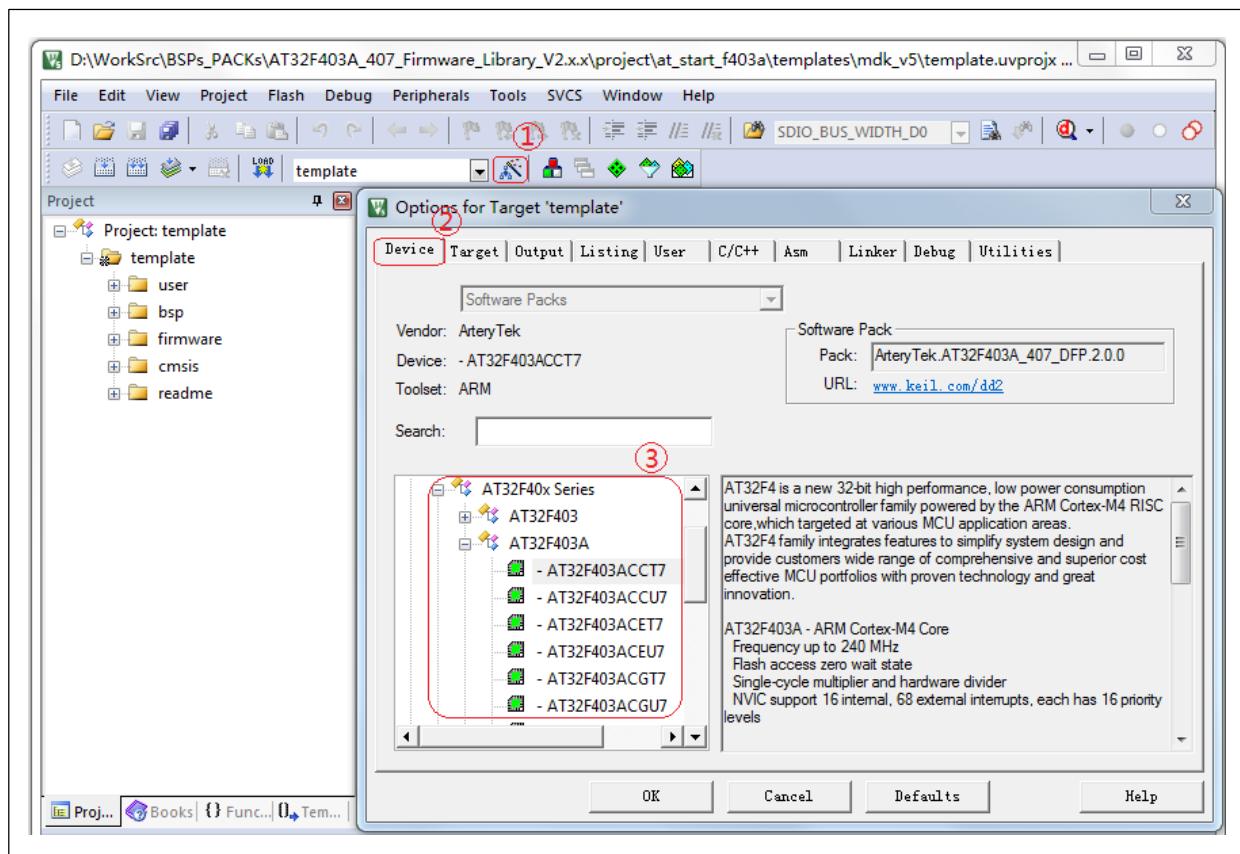


2.2 Keil_v5 Pack installation

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

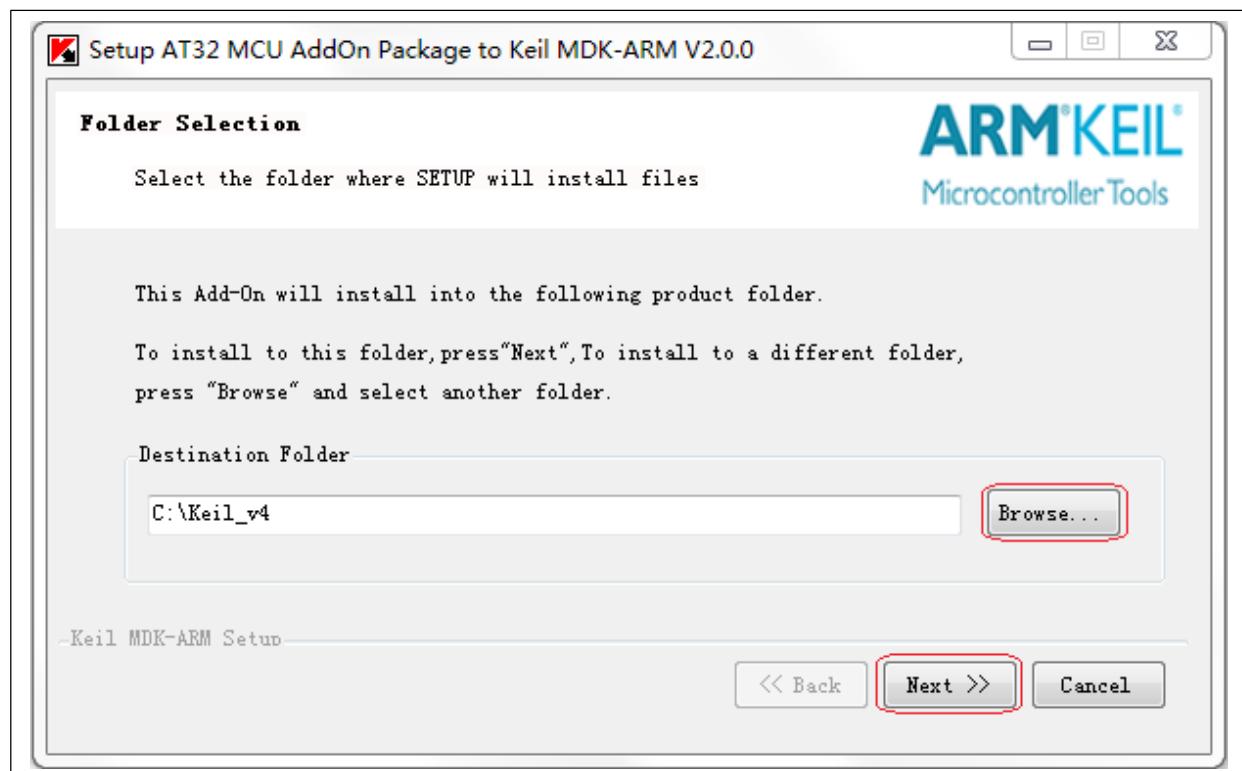


2.3 Keil_v4 Pack installation

Keil4_AT32MCU_AddOn.zip: This is a zip file supporting Keil_v4. Follow the steps below to install:

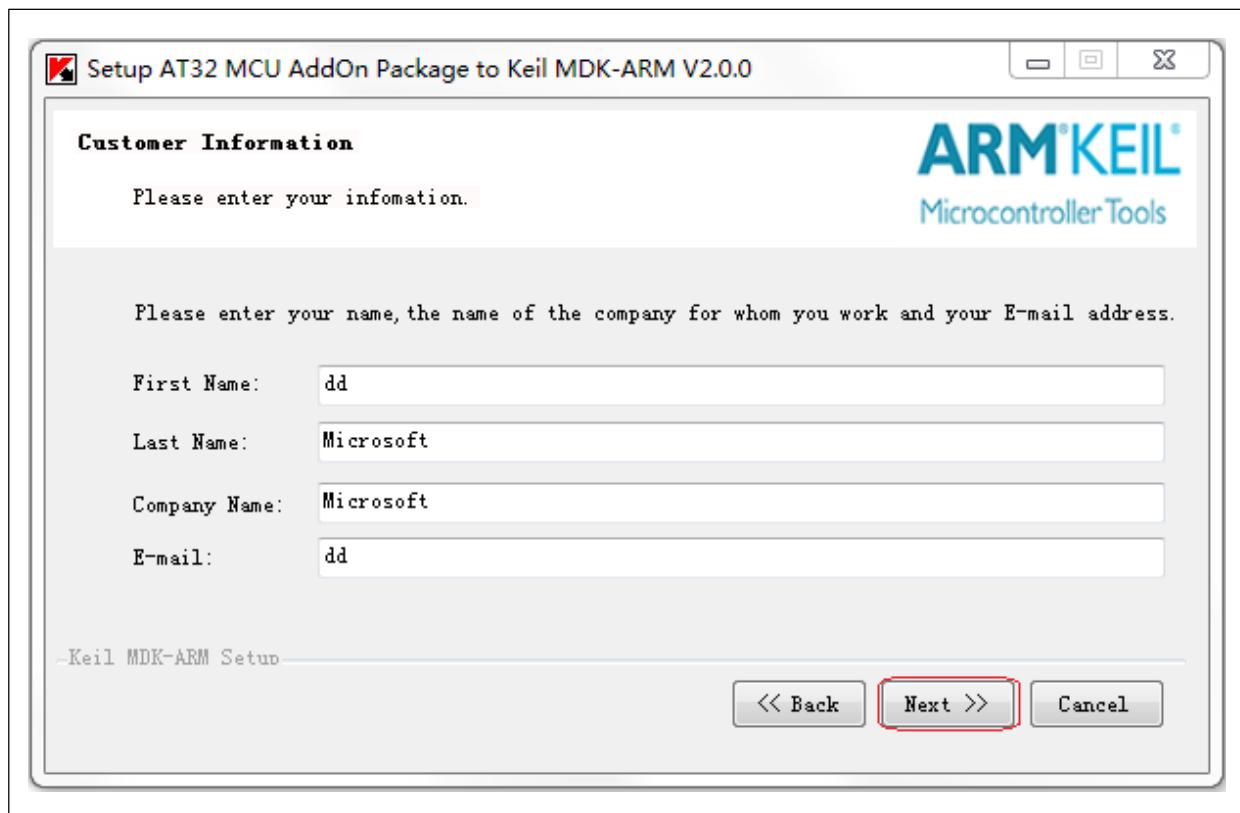
- ① Unzip *Keil4_AT32MCU_AddOn.zip*;
- ② Double click on *Keil4_AT32MCU_AddOn.exe*, and a dialog box pops up below (the specific version information is subject to the actual conditions).

Figure 6. Keil_v4 Pack installation



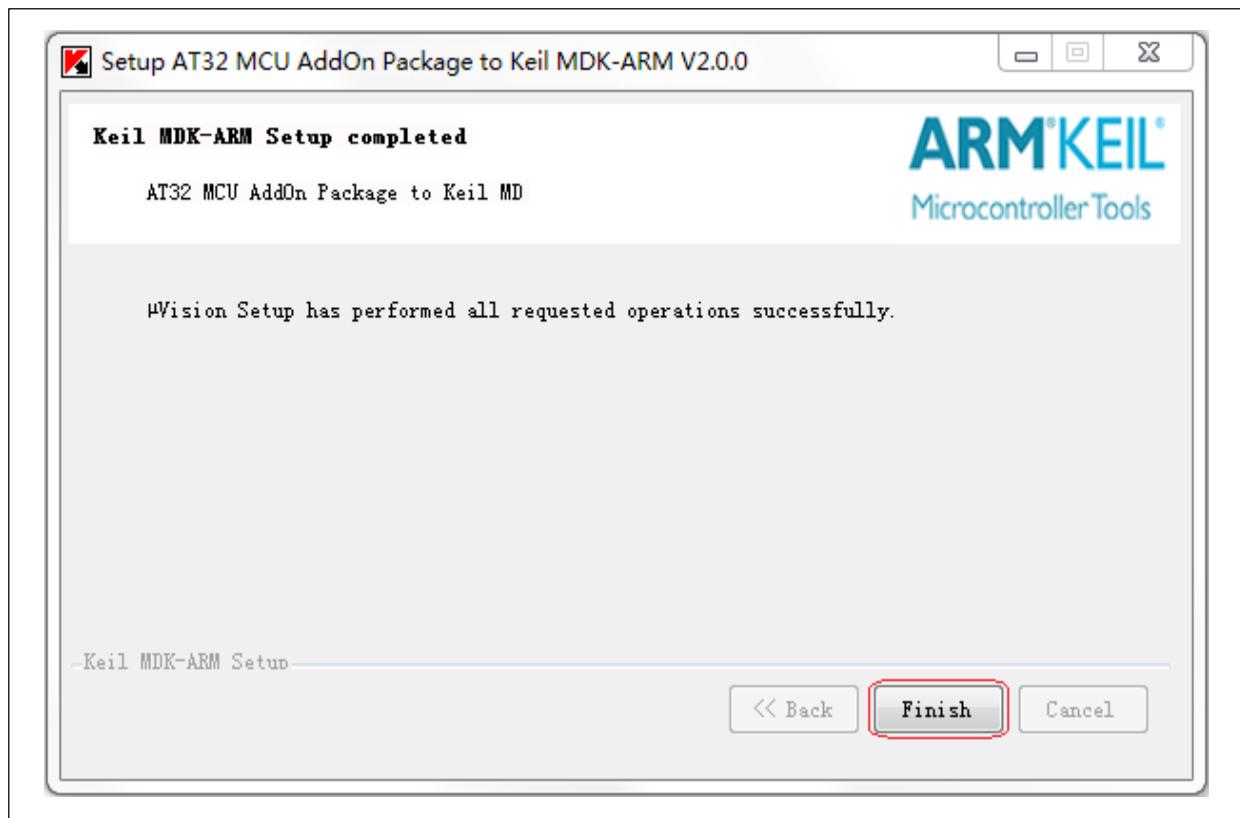
- ③ If the installation path of Keil_v4 does not match the “Destination Folder”, click on “Browse” to select the actual correct path, then click on “Next”, as shown below.

Figure 7. Keil_v4 Pack installation process



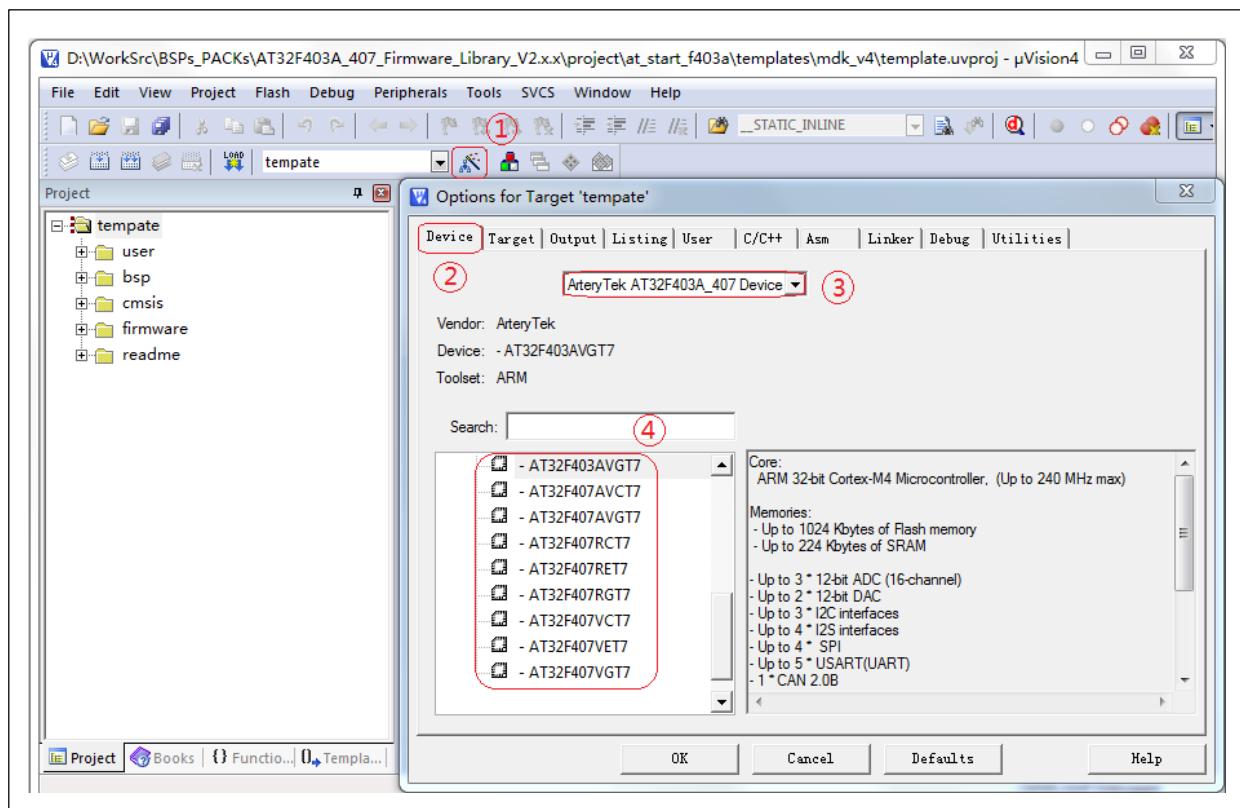
- ④ In the above “Customer Information” window, you can make some changes, but usually it is unnecessary. Then click on “Next” to start installation. The installation result is as follows.

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

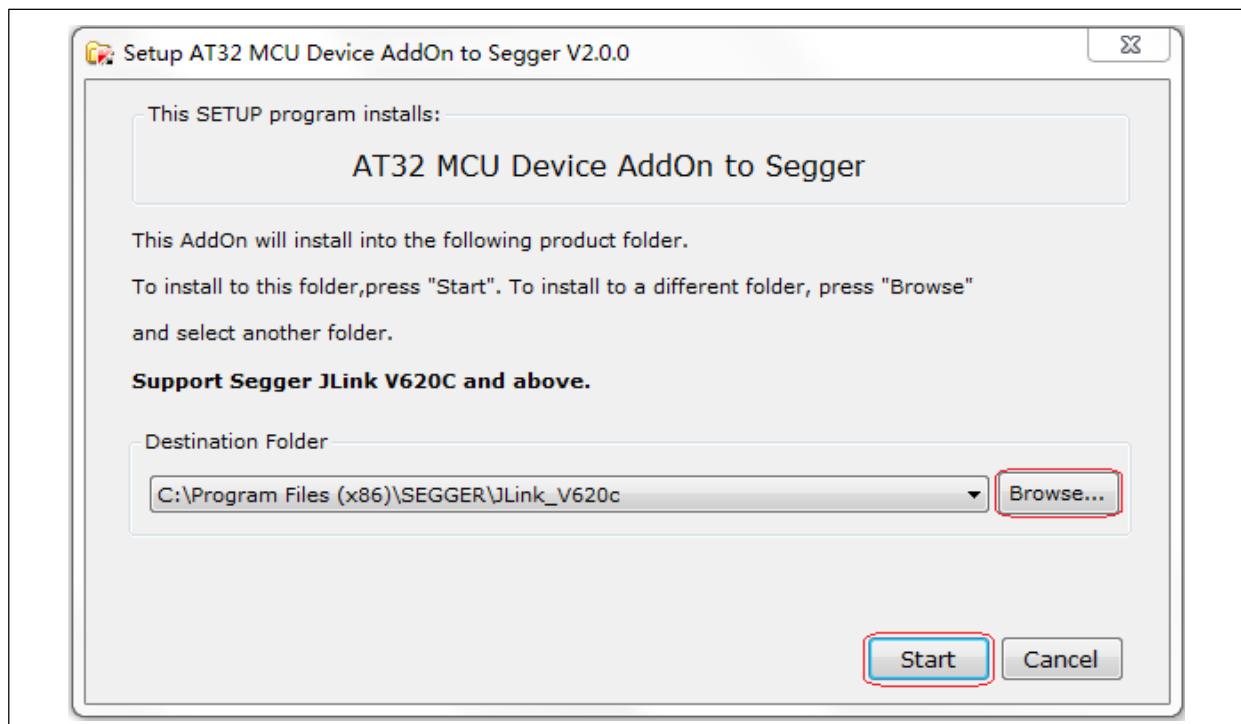


2.4 Segger Pack installation

Segger_AT32MCU_AddOn.zip: This is used to download J-Flash. Follow the steps below to install:

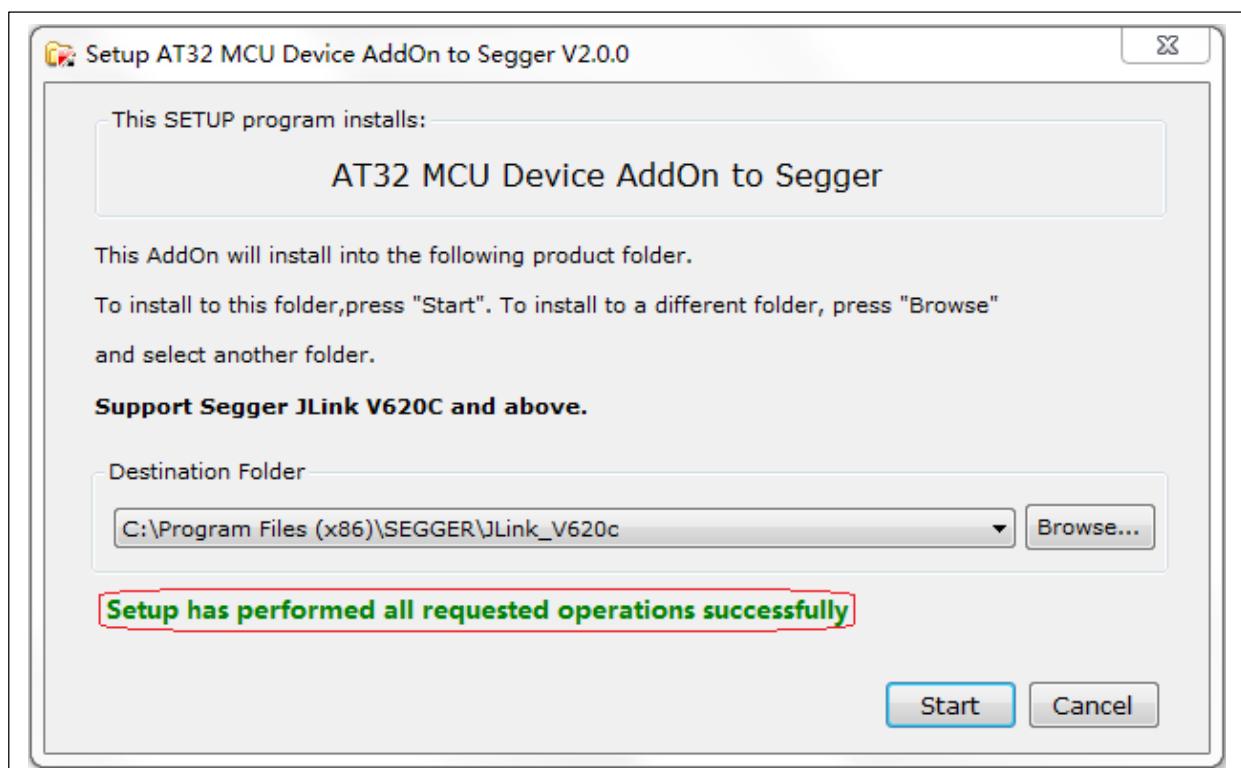
- ① Unzip Segger_AT32MCU_AddOn.zip;
- ② Double click on Segger_AT32MCU_AddOn.exe, and a dialog box pops up below (the specific version information is subject to the actual conditions).

Figure 10. Segger pack installation window



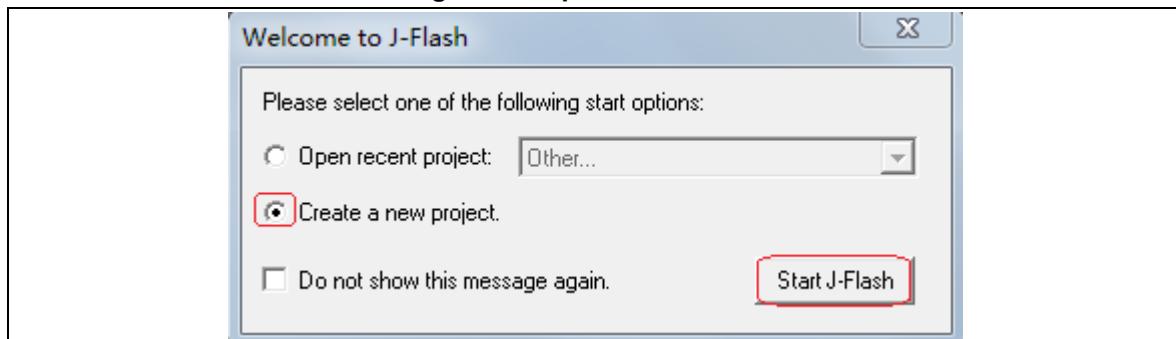
Note: If the installation path of Segger does not match the “Destination Folder”, click on “Browse” to select a correct path, then click on “Start”, as shown below.

Figure 11. Segger pack installation process



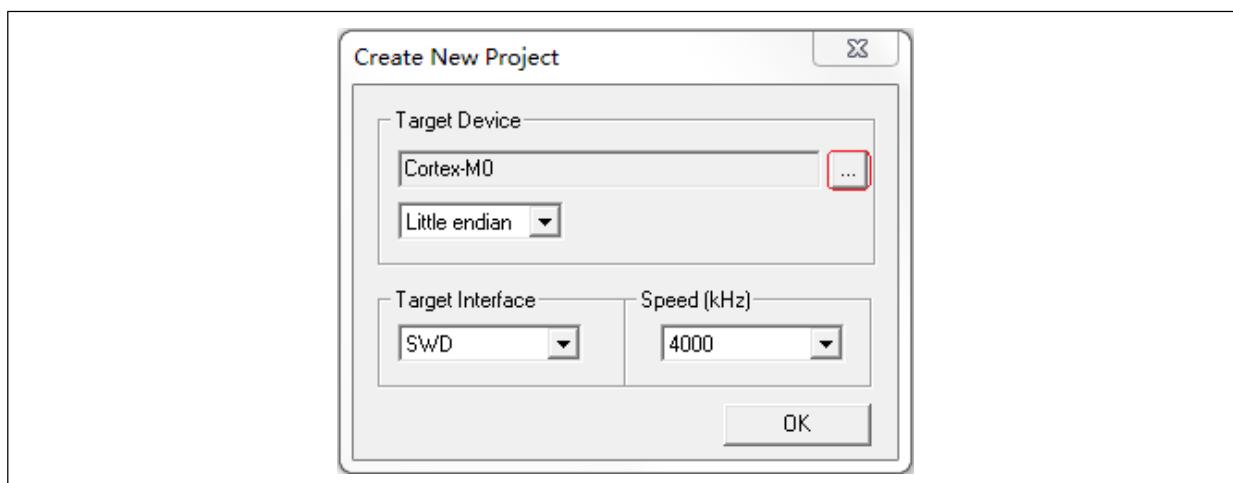
- ③ If the “Setup has performed all requested operations successfully” appears, it indicates successful installation. To check whether the installation is successful or not, follow the steps below:
- Open J-Flash.exe, a dialog box appears; tick “Create a new project” and click on “Start J-Flash”:

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

Select device					
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_FXT_TYPE2_RFAMP	Cortex-M4	2048 KB	224 KB	

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.

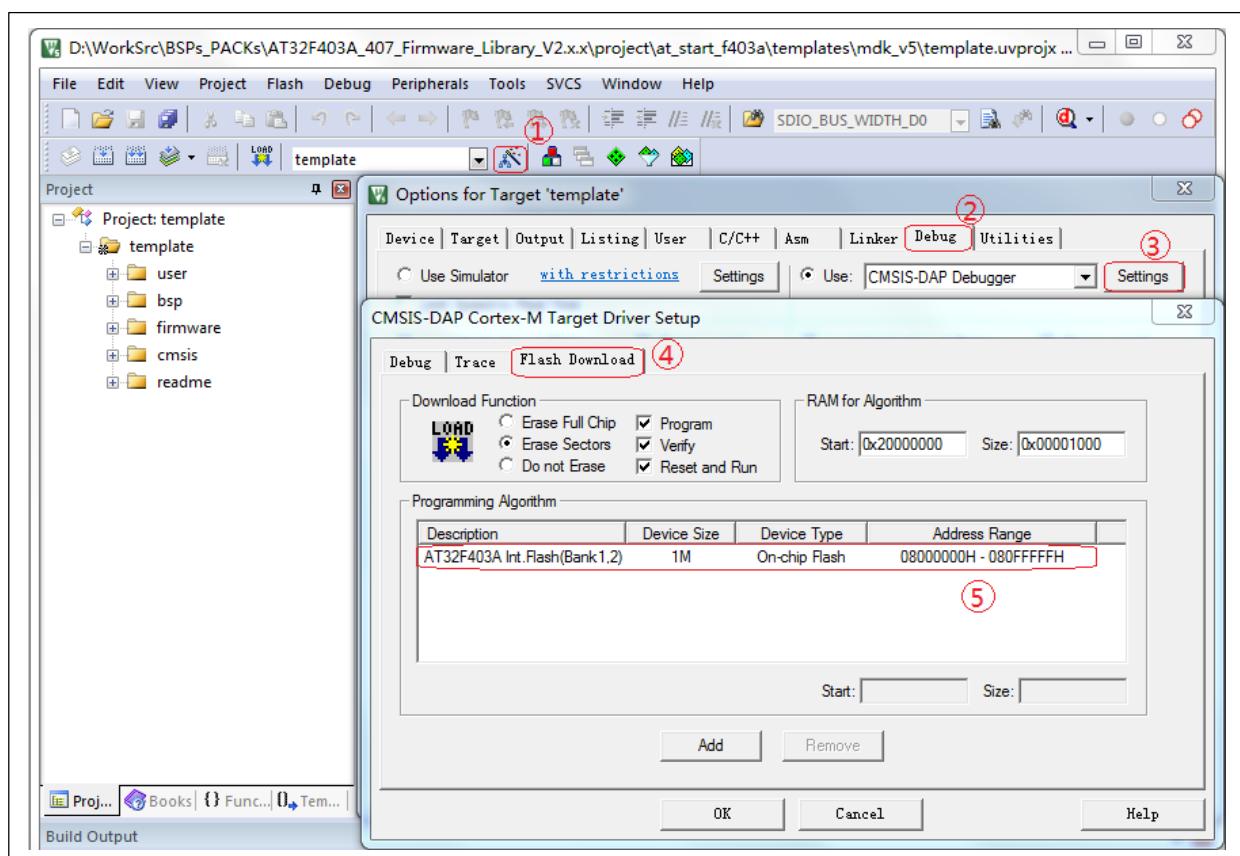
Note: AT32 MCUs have similar Flash algorithms, and this section uses AT32F403A as an example.

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.

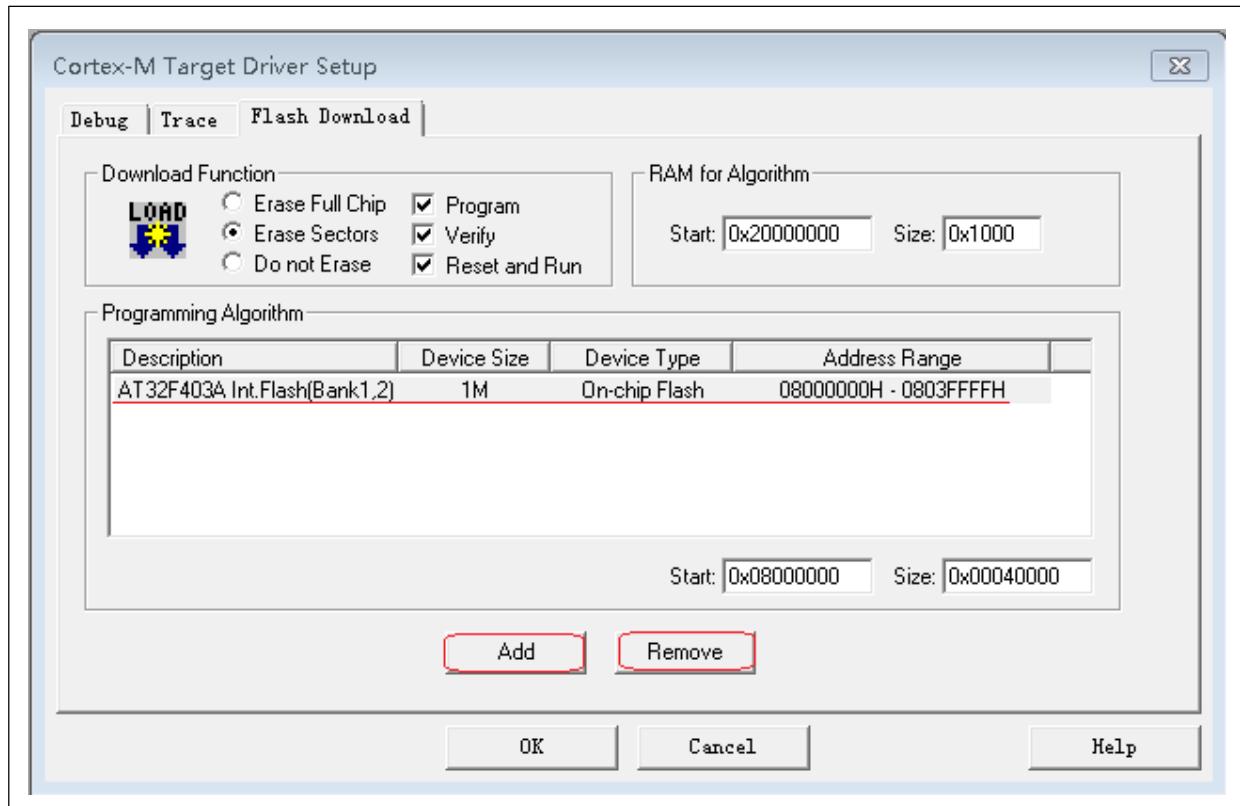
After creating a Keil IDE development tool project, the user can start Debug configuration and select the Flash algorithms. Go to *wand*—>*Debug*—>*Settings*—>*Flash Download*, as shown below:

Figure 15. Keil algorithm file settings



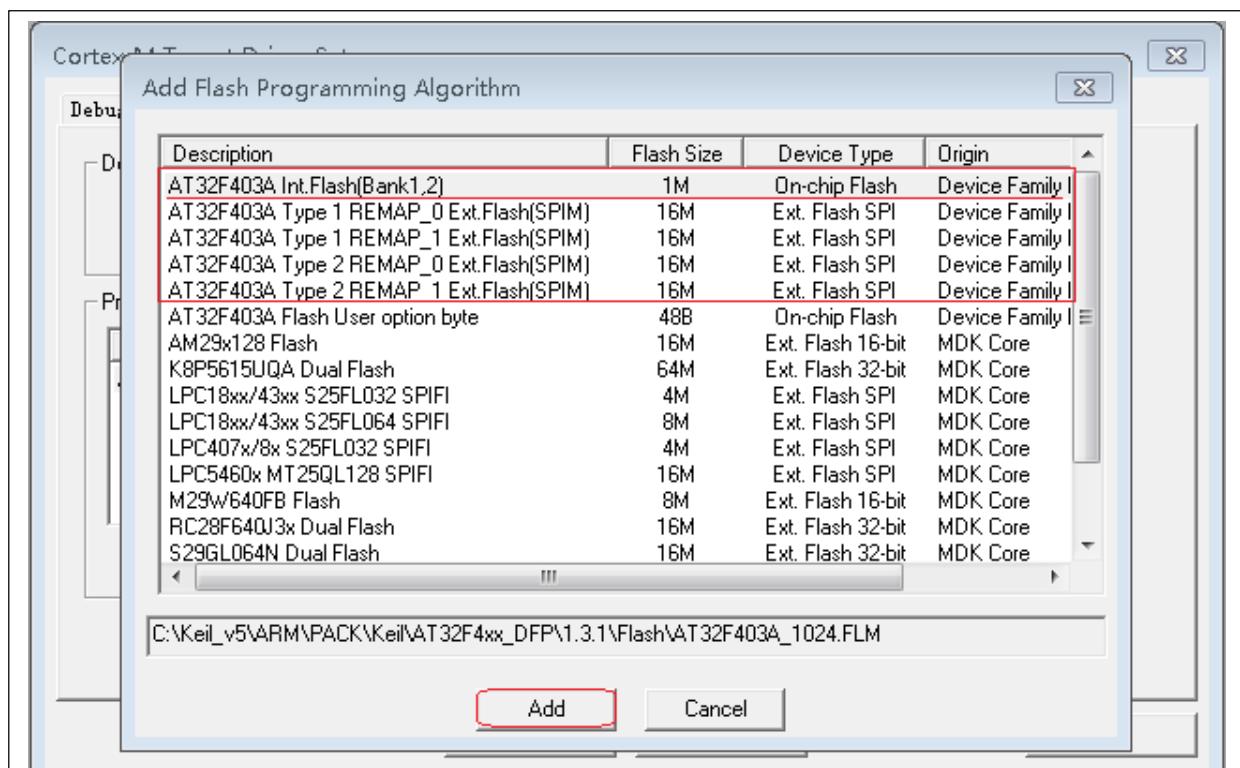
In this example, the selected Flash algorithm file is the default one. To change or remove it, click on this algorithm file, then click on *Add* or *Remove*. If the selected algorithm does not match the MCU, please follow the method below to modify.

Figure 16. Keil algorithm file configuration



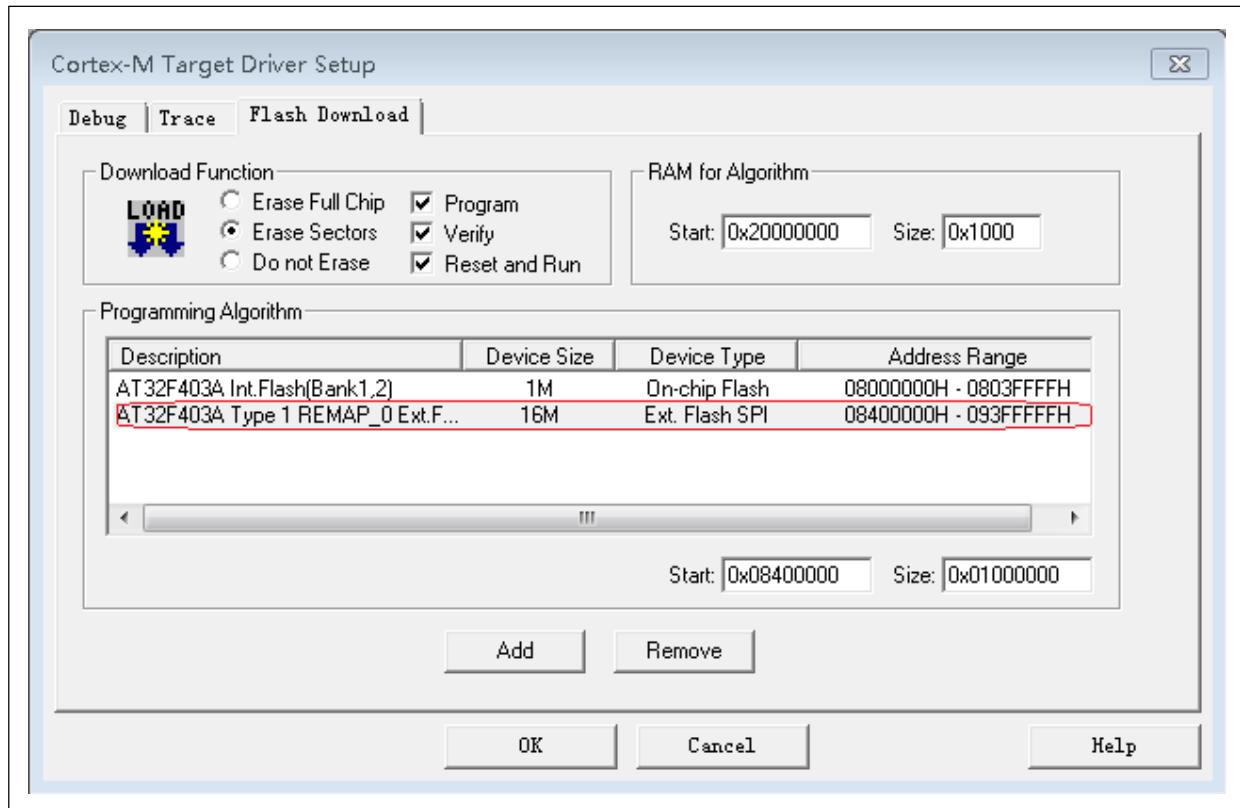
Click on *Remove* to remove the existing algorithm from the configuration, then click on *Add* to view the algorithm files associated with a MCU model and select them, as shown below:

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.

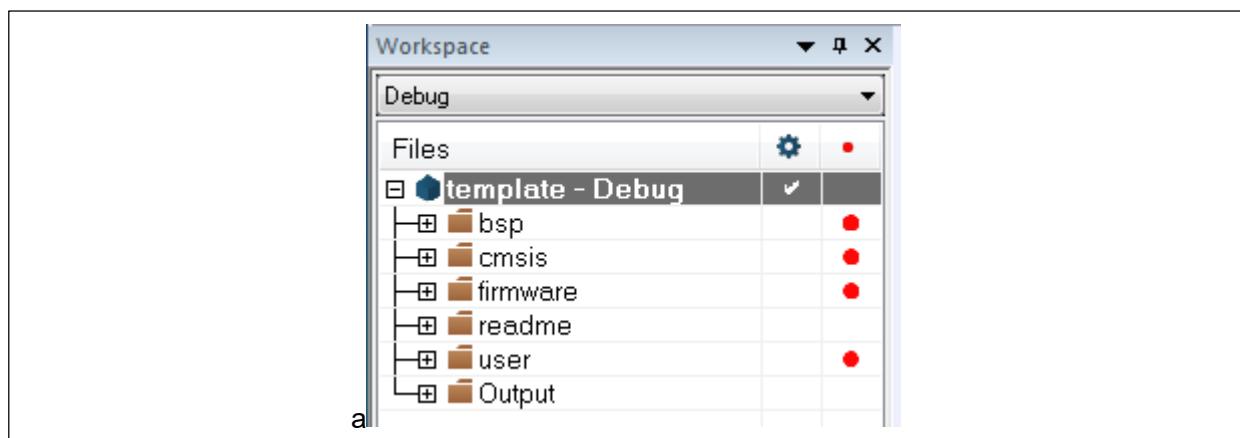
Figure 18. Add algorithm files using Keil



3.2 How to use IAR algorithm files

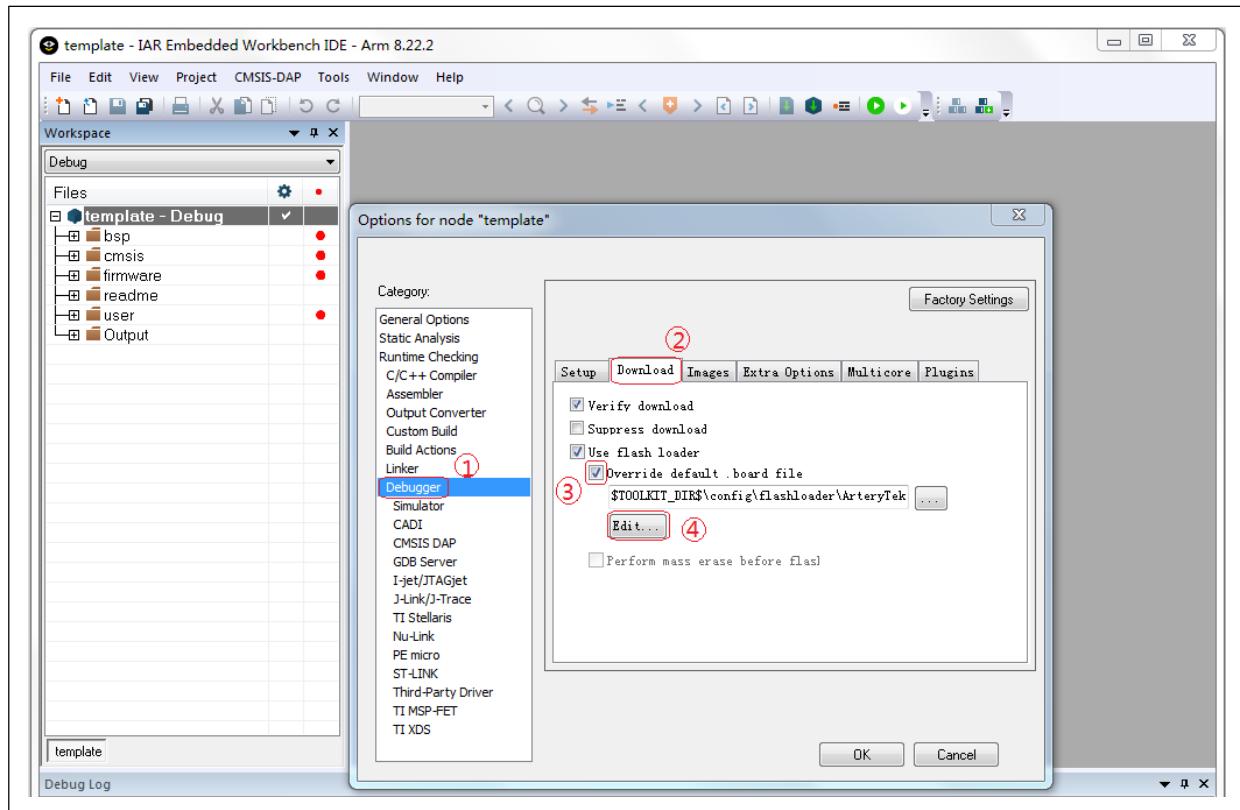
In IAR environment, the Flash algorithm files are automatically selected according to the selected MCU model during a new project configuration. To configure/modify an algorithm file manually, right-click on the file name (after an IAR project is created) in the following gray box:

Figure 19. IAR project name



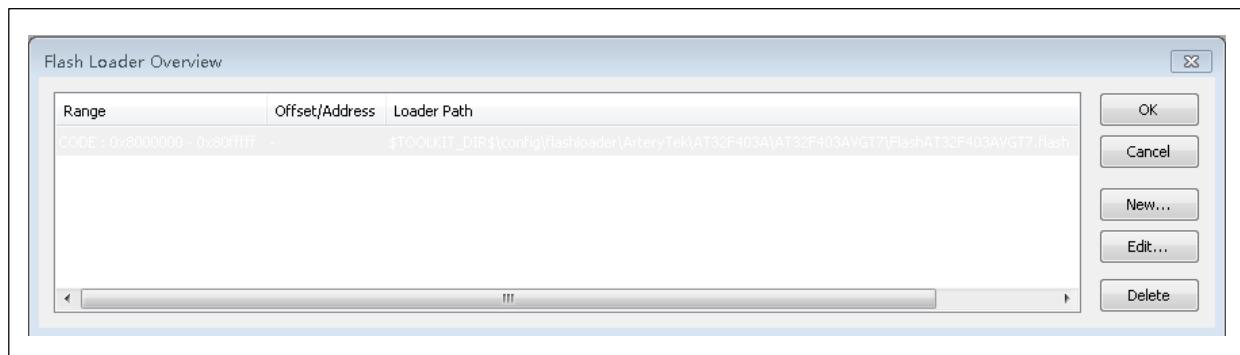
Go to Options—>Debugger—>Download—>Tick Override default .board file—>Click on Edit, as shown below:

Figure 20. IAR algorithm file configuration



Then the following window will be displayed.

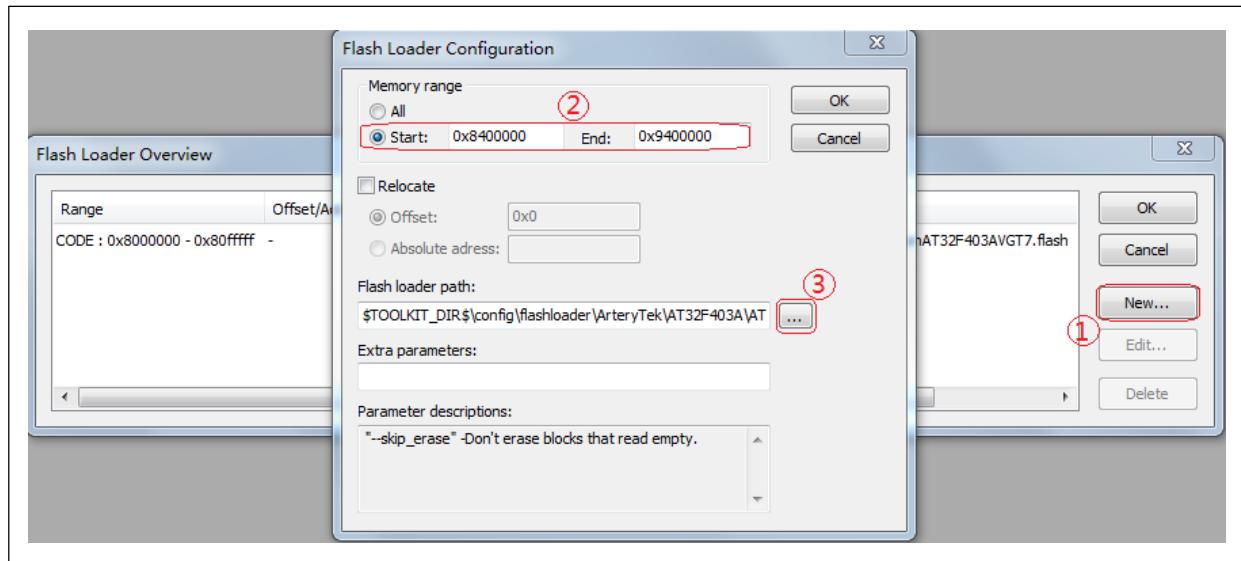
Figure 21. IAR Flash Loader overview



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

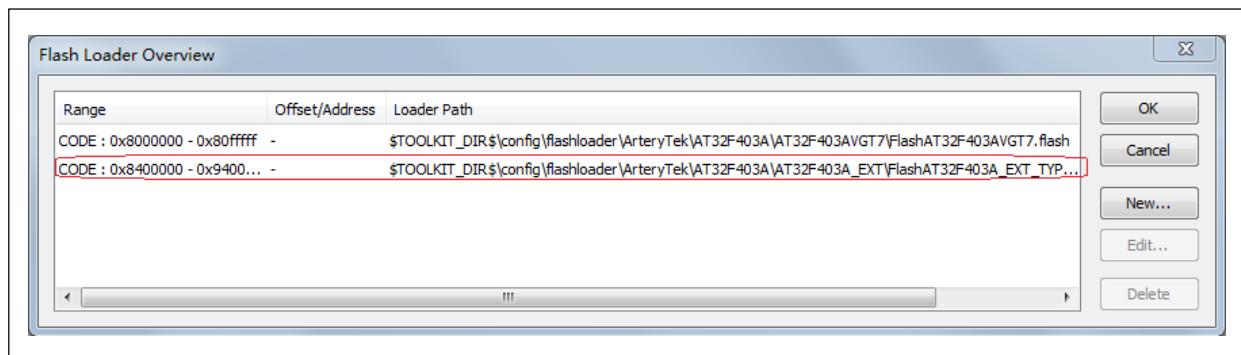
For example, click on *New*—>*Memory range*—> Select a Flash algorithm file, as shown below:

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_AT32MCU_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

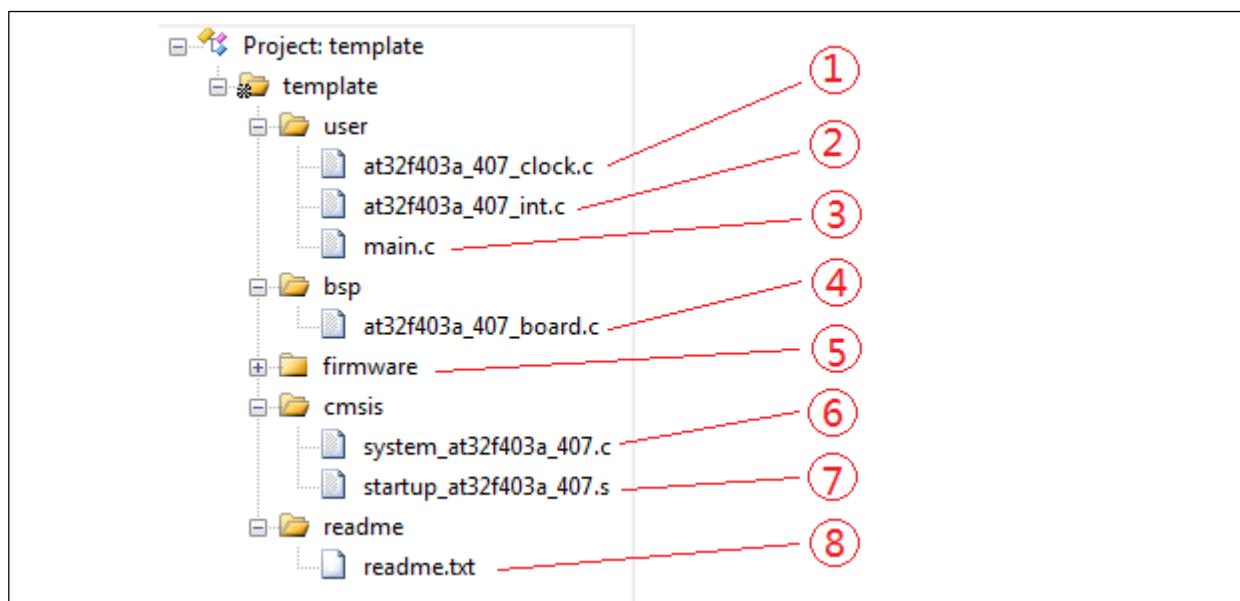
Artery firmware library BSP comes with a series of template projects built around Keil and IAR. For example, the template project of AT32F403A/407 is located in `AT32F403A_407_Firmware_Library_V2.x.x/project/at_start_xxx/templates`.

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, respectively. 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 PWC_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 WWDT_MODULE_ENABLED  
#define WDT_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 EMAC_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 abbreviatio_attribute_action**” or “**peripheral abbreviation_action**”.

The commonly used functions are as follows:

Function type	Naming rule	Example
Peripheral reset	ip_reset	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 header 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 projects, including Keil4, keil5, IAR6, IAR7, IAR8 and eclipse_gcc

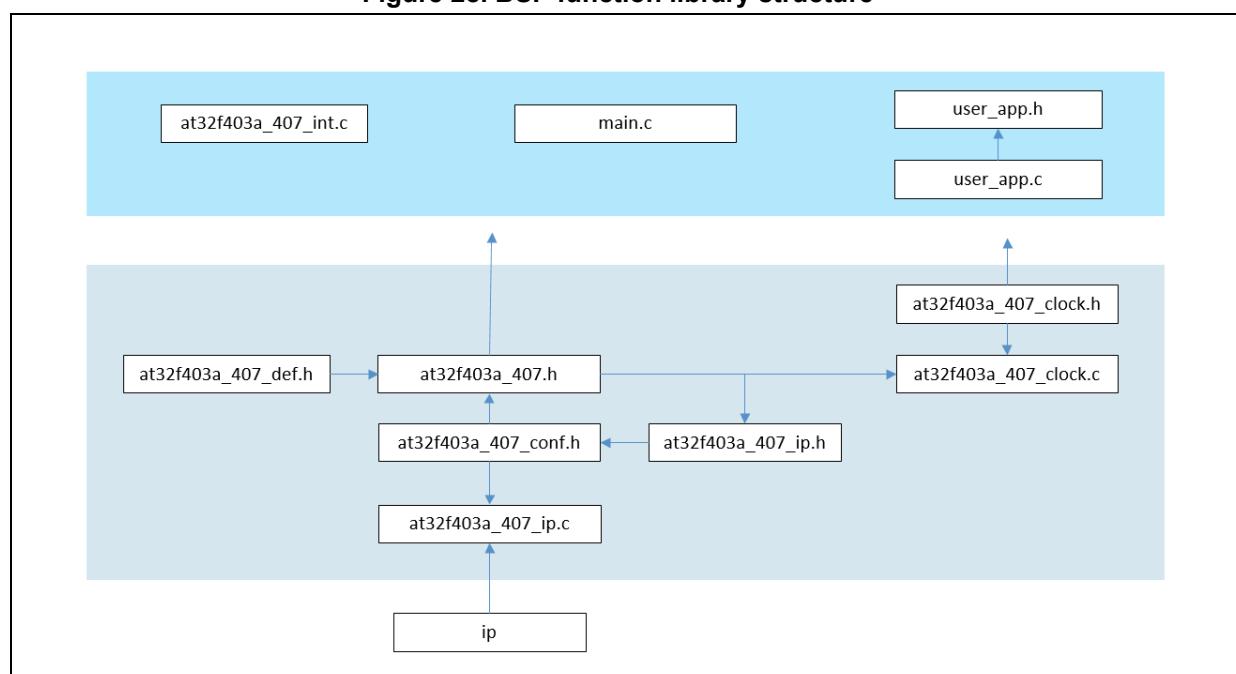
Utilities:

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 configure 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 a 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 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 format description

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 AT32F425 peripheral library functions

### 5.1 HICK automatic clock calibration (ACC)

The ACC register structure acc\_type is defined in the “at32f425\_acc.h”:

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

} acc_type;
```

The table below gives a list of the ACC registers.

**Table 5. Summary of ACC registers**

| Register  | Description            |
|-----------|------------------------|
| acc_sts   | ACC status register    |
| acc_ctrl1 | ACC control register 1 |
| acc_ctrl2 | ACC control register 2 |
| acc_c1    | ACC compare value 1    |
| acc_c2    | ACC compare value 2    |
| acc_c3    | ACC compare value 3    |

The table below gives a list of the ACC library functions.

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

| Function name               | Description                           |
|-----------------------------|---------------------------------------|
| acc_calibration_mode_enable | ACC calibration mode enable           |
| acc_step_set                | Configure ACC calibration step length |
| acc_interrupt_enable        | ACC interrupt enable                  |
| acc_hicktrim_get            | Get ACC trimming calibration value    |
| acc_hickcal_get             | Get ACC coarse calibration value      |
| acc_write_c1                | Write ACC C1 register value           |
| acc_write_c2                | Write ACC C2 register value           |
| acc_write_c3                | Write ACC C3 register value           |
| acc_read_c1                 | Read ACC C1 register value            |
| acc_read_c2                 | Read ACC C2 register value            |
| acc_read_c3                 | Read ACC C3 register value            |
| acc_flag_get                | Get ACC interrupt flag                |
| acc_flag_clear              | Clear ACC interrupt flag              |

## 5.1.1 acc\_calibration\_mode\_enable function

The table below describes the function acc\_calibration\_mode\_enable.

**Table 7. acc\_calibration\_mode\_enable function**

| Name                   | Description                                                                                        |
|------------------------|----------------------------------------------------------------------------------------------------|
| Function name          | acc_calibration_mode_enable                                                                        |
| Function prototype     | void acc_calibration_mode_enable(uint16_t acc_trim, confirm_state new_state);                      |
| Function description   | ACC calibration mode enable                                                                        |
| Input parameter 1      | acc_trim: calibration mode selection<br>This parameter can be ACC_CAL_HICKCAL or ACC_CAL_HICKTRIM. |
| Input parameter 2      | new_state: Enable or disable ACC                                                                   |
| Output parameter       | NA                                                                                                 |
| Return value           | NA                                                                                                 |
| Required preconditions | NA                                                                                                 |
| Called functions       | NA                                                                                                 |

### acc\_trim

Calibration mode selection

ACC\_CAL\_HICKCAL: Coarse calibration mode

ACC\_CAL\_HICKTRIM: Fine calibration mode

### new\_state

Enable or disable ACC

FALSE: Disabled

TRUE: Enabled

### Example:

```
/* open acc calibration */
acc_calibration_mode_enable(ACC_CAL_HICKTRIM, TRUE);
```

## 5.1.2 acc\_step\_set function

The table below describes the function acc\_step\_set.

**Table 8. acc\_step\_set function**

| Name                   | Description                            |
|------------------------|----------------------------------------|
| Function name          | acc_step_set                           |
| Function prototype     | void acc_step_set(uint8_t step_value); |
| Function description   | Configure ACC calibration step length  |
| Input parameter 1      | step_value: step value for calibration |
| Output parameter       | NA                                     |
| Return value           | NA                                     |
| Required preconditions | NA                                     |
| Called functions       | NA                                     |

### step\_value

This 4-bit field defines the value to be changed for each calibration.

Note: To obtain better calibration accuracy, it is recommended to set the step value to 1.

When ENTRIM=0 only the HICKCAL is calibrated. If the step value is incremented or decremented by one, the corresponding HICKCAL follows the change rule (increased or decreased by one), and the HICK frequency will increase or decrease by 40 KHz (design value), meaning a positive

correlation between them.

When ENTRIM=1, only the HICKTRIM is calibrated. If the step value is incremented or decremented by one, the corresponding HICKTRIM follows the change rule (increased or decreased by one), and the HICK frequency will increase or decrease by 20KHz (design value), meaning a positive correlation between them.

**Example:**

```
/* set acc step value */
acc_step_set(0x1);
```

### 5.1.3 acc\_interrupt\_enable function

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

**Table 9. acc\_interrupt\_enable function**

| Name                   | Description                                                           |
|------------------------|-----------------------------------------------------------------------|
| Function name          | dma_interrupt_enable                                                  |
| Function prototype     | void acc_interrupt_enable(uint16_t acc_int, confirm_state new_state); |
| Function description   | Enable acc interrupts                                                 |
| Input parameter 1      | acc_int: interrupt source selection                                   |
| Input parameter 2      | new_state: enable or disable interrupts                               |
| Output parameter       | NA                                                                    |
| Return value           | NA                                                                    |
| Required preconditions | NA                                                                    |
| Called functions       | NA                                                                    |

**acc\_int**

interrupt source selection

ACC\_CALRDYIEN\_INT: Calibration complete interrupt

ACC\_EIEN\_INT: Reference signal lost interrupt

**new\_state**

Enable or disable interrupts.

FALSE: Interrupt disabled

TRUE: Interrupt enabled

**Example:**

```
/* enable the acc reference signal lost interrupt */
acc_interrupt_enable(ACC_EIEN_INT, TRUE);
```

## 5.1.4 acc\_hicktrim\_get function

The table below describes the function acc\_hicktrim\_get.

**Table 10. acc\_hicktrim\_get function**

| Name                   | Description                     |
|------------------------|---------------------------------|
| Function name          | acc_hicktrim_get                |
| Function prototype     | uint8_t acc_hicktrim_get(void); |
| Function description   | Get ACC trimming value          |
| Input parameter        | NA                              |
| Output parameter       | NA                              |
| Return value           | Return ACC trimming value       |
| Required preconditions | NA                              |
| Called functions       | NA                              |

**Example:**

```
/* get trim value*/
uint8_t trim_value;
trim_value = acc_hicktrim_get();
```

## 5.1.5 acc\_hickcal\_get function

The table below describes the function acc\_hickcal\_get.

**Table 11. acc\_hickcal\_get function**

| Name                   | Description                         |
|------------------------|-------------------------------------|
| Function name          | acc_hickcal_get                     |
| Function prototype     | uint8_t acc_hickcal_get(void);      |
| Function description   | Get ACC coarse calibration value    |
| Input parameter        | NA                                  |
| Output parameter       | NA                                  |
| Return value           | Return ACC coarse calibration value |
| Required preconditions | NA                                  |
| Called functions       | NA                                  |

**Example:**

```
/* get cal value*/
uint8_t cal_value;
cal_value = acc_hickcal_get();
```

## 5.1.6 acc\_write\_c1 function

The table below describes the function acc\_write\_c1.

**Table 12. acc\_write\_c1 function**

| Name                   | Description                                              |
|------------------------|----------------------------------------------------------|
| Function name          | acc_write_c1                                             |
| Function prototype     | void acc_write_c1(uint16_t acc_c1_value);                |
| Function description   | Write ACC C1 register value                              |
| Input parameter        | acc_c1_value: the value to be written in ACC C1 register |
| Output parameter       | NA                                                       |
| Return value           | NA                                                       |
| Required preconditions | NA                                                       |
| Called functions       | NA                                                       |

**Example:**

```
/* update the c1 value */
acc_c2_value = 8000;
acc_write_c1(acc_c2_value - 10);
```

## 5.1.7 acc\_write\_c2 function

The table below describes the function acc\_write\_c2.

**Table 13. acc\_write\_c2 function**

| Name                   | Description                                              |
|------------------------|----------------------------------------------------------|
| Function name          | acc_write_c2                                             |
| Function prototype     | void acc_write_c2(uint16_t acc_c2_value);                |
| Function overview      | Write ACC C2 register value                              |
| Input parameter        | acc_c2_value: the value to be written in ACC C2 register |
| Output parameter       | NA                                                       |
| Return value           | NA                                                       |
| Required preconditions | NA                                                       |
| Called functions       | NA                                                       |

**Example:**

```
/* update the c2 value */
acc_c2_value = 8000;
acc_write_c2(acc_c2_value - 10);
```

## 5.1.8 acc\_write\_c3 function

The table below describes the function acc\_write\_c3.

**Table 14. acc\_write\_c3 function**

| Name                   | Description                                              |
|------------------------|----------------------------------------------------------|
| Function name          | acc_write_c3                                             |
| Function prototype     | void acc_write_c3(uint16_t acc_c3_value);                |
| Function description   | Write ACC C3 register value                              |
| Input parameter        | acc_c3_value: the value to be written in ACC C3 register |
| Output parameter       | NA                                                       |
| Return value           | NA                                                       |
| Required preconditions | NA                                                       |
| Called functions       | NA                                                       |

**Example:**

```
/* update the c3 value */
acc_c2_value = 8000;
acc_write_c3(acc_c2_value - 10);
```

## 5.1.9 acc\_read\_c1 function

The table below describes the function acc\_read\_c1.

**Table 15. acc\_read\_c1 function**

| Name                   | Description                 |
|------------------------|-----------------------------|
| Function name          | acc_read_c1                 |
| Function prototype     | uint16_t acc_read_c1(void); |
| Function description   | Read ACC C1 register value  |
| Input parameter        | NA                          |
| Output parameter       | NA                          |
| Return value           | ACC C1 register value       |
| Required preconditions | NA                          |
| Called functions       | NA                          |

**Example:**

```
/* get the c1 value */
uint16_t acc_c1_value;
acc_c1_value = acc_read_c1();
```

## 5.1.10 acc\_read\_c2 function

The table below describes the function acc\_read\_c2.

**Table 16. acc\_read\_c2 function**

| Name                   | Description                 |
|------------------------|-----------------------------|
| Function name          | acc_read_c2                 |
| Function prototype     | uint16_t acc_read_c2(void); |
| Function description   | Read ACC C2 register value  |
| Input parameter        | NA                          |
| Output parameter       | NA                          |
| Return value           | ACC C2 register value       |
| Required preconditions | NA                          |
| Called functions       | NA                          |

**Example:**

```
/* get the c2 value */
uint16_t acc_c2_value;
acc_c2_value = acc_read_c2();
```

## 5.1.11 acc\_read\_c3 function

The table below describes the function acc\_read\_c3.

**Table 17. acc\_read\_c3 function**

| Name                   | Description                 |
|------------------------|-----------------------------|
| Function name          | acc_read_c3                 |
| Function prototype     | uint16_t acc_read_c3(void); |
| Function description   | Read ACC C3 register value  |
| Input parameter        | NA                          |
| Output parameter       | NA                          |
| Return value           | ACC C3 register value       |
| Required preconditions | NA                          |
| Called functions       | NA                          |

**Example:**

```
/* get the c3 value */
uint16_t acc_c3_value;
acc_c3_value = acc_read_c3();
```

## 5.1.12 acc\_flag\_get function

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

Table 18. acc\_flag\_get function

| Name                   | Description                                                 |
|------------------------|-------------------------------------------------------------|
| Function name          | acc_flag_get                                                |
| Function prototype     | flag_status acc_flag_get(uint16_t acc_flag);                |
| Function description   | Get ACC flag status                                         |
| Input parameter 1      | acc_flag: ACC flag selection                                |
| Output parameter       | NA                                                          |
| Return value           | flag_status: indicates whether or not the flag has been set |
| Required preconditions | NA                                                          |
| Called functions       | NA                                                          |

### acc\_flag

The acc\_flag is used for flag selection, including:

ACC\_RSLOST\_FLAG: Reference signal lost interrupt

ACC\_CALRDY\_FLAG: Calibration complete interrupt

### flag\_status

RESET: Corresponding flag bit is not set

SET: Corresponding flag bit is set

### Example:

```
if(acc_flag_get(ACC_CALRDY_FLAG) != RESET)
{
 at32_led_toggle(LED2);
 /* clear acc calibration ready flag */
 acc_flag_clear(ACC_CALRDY_FLAG);
}
```

## 5.1.13 acc\_interrupt\_flag\_get function

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

**Table 19. acc\_flag\_clear function**

| Name                   | Description                                            |
|------------------------|--------------------------------------------------------|
| Function name          | acc_interrupt_flag_get                                 |
| Function prototype     | flag_status acc_interrupt_flag_get(uint16_t acc_flag); |
| Function description   | Get ACC interrupt flag status                          |
| Input parameter 1      | acc_flag: ACC flag selection                           |
| Output parameter       | NA                                                     |
| Return value           | flag_status: SET or RESET                              |
| Required preconditions | NA                                                     |
| Called functions       | NA                                                     |

### acc\_flag

The acc\_flag is used for flag selection, including:

ACC\_RSLOST\_FLAG: Reference signal lost interrupt

ACC\_CALRDY\_FLAG: Calibration complete interrupt

### flag\_status

RESET: Corresponding flag bit is not set

SET: Corresponding flag bit is set

### Example:

```
if(acc_interrupt_flag_get(ACC_CALRDY_FLAG) != RESET)
{
 at32_led_toggle(LED2);
 /* clear acc calibration ready flag */
 acc_flag_clear(ACC_CALRDY_FLAG);
}
```

## 5.1.14 acc\_flag\_clear function

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

**Table 20. acc\_flag\_clear function**

| Name                   | Description                             |
|------------------------|-----------------------------------------|
| Function name          | acc_flag_clear                          |
| Function prototype     | void acc_flag_clear(uint16_t acc_flag); |
| Function description   | Clear ACC flag                          |
| Input parameter 1      | acc_flag: ACC flag selection            |
| Output parameter       | NA                                      |
| Return value           | NA                                      |
| Required preconditions | NA                                      |
| Called functions       | NA                                      |

### acc\_flag

The acc\_flag is used for flag selection, including:

ACC\_RSLOST\_FLAG: Reference signal lost interrupt

ACC\_CALRDY\_FLAG: Calibration complete interrupt

**Example:**

```
if(acc_flag_get(ACC_CALRDY_FLAG) != RESET)
{
 at32_led_toggle(LED2);
 /* clear acc calibration ready flag */
 acc_flag_clear(ACC_CALRDY_FLAG);
}
```

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

ADC register structure adc\_type is defined in the “at32f425\_adc.h”.

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

} adc_type;
```

The table below gives a list of the ADC registers.

**Table 21. 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 22. 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_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_preempt_conversion_data_get           | Get the converted data of preempted group                                                               |
| adc_flag_get                              | Get the status of flag bits                                                                             |
| adc_flag_clear                            | Clear flag bits                                                                                         |

## 5.2.1 adc\_reset function

The table below describes the function adc\_reset.

**Table 23. 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: the selected ADC peripheral<br>This parameter can be ADC1. |
| Output parameter       | NA                                                                |
| Return value           | NA                                                                |
| Required preconditions | NA                                                                |
| Called functions       | crm_periph_reset()                                                |

**Example:**

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

## 5.2.2 adc\_enable function

The table below describes the function adc\_enable.

**Table 24. 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<br>This parameter is used to select ADC1.                             |
| Input parameter 2      | new_state: indicates the pre-configured status of A/D converter<br>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 adc\_enable function while ADC is enabled will trigger ordinary channels conversion.*

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

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

**Table 25. 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 value for the adc_base_struct.                         |
| Input parameter        | adc_base_struct: adc_base_config_type pointer                          |
| Output parameter       | NA                                                                     |
| Return value           | NA                                                                     |
| Required preconditions | NA                                                                     |
| Called functions       | NA                                                                     |

The default values of members in the adc\_base\_struct:

|                          |                     |
|--------------------------|---------------------|
| 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);
```

### 5.2.4 adc\_base\_config function

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

**Table 26. 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<br>This parameter can be ADC1.    |
| 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 at32f425\_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);

```

## 5.2.5 adc\_dma\_mode\_enable function

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

**Table 27. 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<br>This parameter can be ADC1.                                     |
| Input parameter 2      | new_state: pre-configured status of ordinary group in DMA transfer mode<br>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);

```

## 5.2.6 adc\_interrupt\_enable function

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

**Table 28. 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<br>This parameter can be ADC1.                                      |
| Input parameter 2      | adc_int: ADC event interrupt selection<br>This parameter is used to select any event interrupt supported by ADC. |
| Input parameter3       | new_state: indicates the pre-configured status of ADC event interrupts<br>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: Channel conversion end interrupt enable

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);
```

## 5.2.7 adc\_calibration\_init function

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

**Table 29. adc\_calibration\_init function**

| Name                   | Description                                                                 |
|------------------------|-----------------------------------------------------------------------------|
| Function name          | adc_calibration_init                                                        |
| Function prototype     | void adc_calibration_init(adc_type *adc_x)                                  |
| Function description   | Initialization calibration                                                  |
| Input parameter        | adc_x: indicates the selected ADC peripheral<br>This parameter can be ADC1. |
| Output parameter       | NA                                                                          |
| Return value           | NA                                                                          |
| Required preconditions | NA                                                                          |
| Called functions       | NA                                                                          |

#### Example:

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

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

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

**Table 30. 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<br>This parameter can be ADC1.             |
| Output parameter       | NA                                                                                      |
| Return value           | flag_status: indicates the status of calibration initialization<br>Return SET or RESET. |
| Required preconditions | NA                                                                                      |
| Called functions       | NA                                                                                      |

**Example:**

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

## 5.2.9 adc\_calibration\_start function

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

**Table 31. 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<br>This parameter can be ADC1. |
| Output parameter       | NA                                                                          |
| Return value           | NA                                                                          |
| Required preconditions | NA                                                                          |
| Called functions       | NA                                                                          |

**Example:**

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

## 5.2.10 adc\_calibration\_status\_get function

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

**Table 32. 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<br>This parameter can be ADC1. |
| Output parameter       | NA                                                                          |
| Return value           | flag_status: indicates the status of calibration<br>Return SET or RESET.    |
| Required preconditions | NA                                                                          |
| Called functions       | NA                                                                          |

**Example:**

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

## 5.2.11 adc\_voltage\_monitor\_enable function

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

**Table 33. 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<br>This parameter can be ADC1.                                                                                                                 |
| Input parameter 2      | adc_voltage_monitoring: select ordinary group, preempted group or a single channel for voltage monitoring<br>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

The 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);
```

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

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

**Table 34. 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<br>This parameter can be ADC1.                                                                                                |
| Input parameter 2      | adc_high_threshold: indicates the upper limit for voltage monitoring<br>This parameter can be any value between 0x000 and 0xFFFF.                                          |
| Input parameter3       | adc_low_threshold: indicates the lower limit for voltage monitoring<br>This parameter can be any value lower than that of 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);
```

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

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

**Table 35. 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,<br>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<br>This parameter can be ADC1.                                      |
| Input parameter 2      | adc_channel: select a single channel for voltage monitoring<br>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);
```

## 5.2.14 adc\_ordinary\_channel\_set function

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

**Table 36. 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<br>adc_channel, uint8_t adc_sequence, adc_sampletime_select_type<br>adc_sampletime) |
| Function description | Configure ordinary channels, including parameters such as channel selection,<br>conversion sequence number and sampling time                               |
| Input parameter 1    | adc_x: indicates the selected ADC peripheral<br>This parameter can be ADC1.                                                                                |
| Input parameter 2    | adc_channel: indicates the selected channel<br>Refer to <a href="#">adc_channel</a> for details.                                                           |
| Input parameter3     | adc_sequence: defines the sequence of channel conversion<br>This parameter can be any value from 1 to 16.                                                  |

| Name                   | Description                                                                                                 |
|------------------------|-------------------------------------------------------------------------------------------------------------|
| Input parameter4       | adc_sampetime: defines the sampling time for channel<br>Refer to <a href="#">adc_sampetime</a> for details. |
| Output parameter       | NA                                                                                                          |
| Return value           | NA                                                                                                          |
| Required preconditions | NA                                                                                                          |
| Called functions       | NA                                                                                                          |

**adc\_sampetime**

The adc\_sampetime 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);
```

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

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

Table 37. 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<br>This parameter can be ADC1.                                                       |
| Input parameter 2      | adc_channel_length: set the length of preempted channel conversion<br>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);
```

## 5.2.16 adc\_preempt\_channel\_set function

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

**Table 38. 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 parameters such as channel selection, conversion sequence number and sampling time                             |
| Input parameter 1      | adc_x: indicates the selected ADC<br>This parameter can be ADC1.                                                                                    |
| Input parameter 2      | adc_channel: indicates the selected channel<br>Refer to <a href="#">adc_channel</a> for details.                                                    |
| Input parameter3       | adc_sequence: set the sequence number for channel conversion<br>This parameter can be any value from 1 to 4.                                        |
| Input parameter4       | adc_sampletime: set the sampling time for channels<br>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);
```

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

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

**Table 39. 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<br>This parameter can be ADC1.                                                                                                  |
| Input parameter 2    | adc_ordinary_trig: indicates the selected trigger event for ordinary group<br>This parameter can be any enumerated value in the<br>adc_ordinary_trig_select_type. |
| Input parameter3     | new_state: the preset status of trigger mode<br>This parameter can be TRUE or FALSE.                                                                              |
| Output parameter     | NA                                                                                                                                                                |
| Return value         | NA                                                                                                                                                                |

| Name                   | Description |
|------------------------|-------------|
| Required preconditions | NA          |
| Called functions       | NA          |

**adc\_ordinary\_trig**

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

|                                  |                        |
|----------------------------------|------------------------|
| ADC12_ORDINARY_TRIG_TMR1TRGOUT:  | TMR1 TRGOUT event      |
| ADC12_ORDINARY_TRIG_TMR1CH4:     | TMR1 CH4 event         |
| ADC12_ORDINARY_TRIG_TMR2TRGOUT:  | TMR2 TRGOUT event      |
| ADC12_ORDINARY_TRIG_TMR3TRGOUT:  | TMR3 TRGOUT event      |
| ADC12_ORDINARY_TRIG_TMR15TRGOUT: | TMR15 TRGOUT event     |
| ADC12_ORDINARY_TRIG_TMR1CH1:     | TMR1 CH1 event         |
| ADC12_ORDINARY_TRIG_EXINT11:     | EXINT 11 event         |
| ADC12_ORDINARY_TRIG_SOFTWARE:    | Software trigger event |

**Example:**

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

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

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

**Table 40. 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,<br>adc_preempt_trig_select_type adc_preempt_trig, confirm_state new_state)                              |
| Function description   | Enable trigger mode and trigger event selection for preempted group conversion                                                                                   |
| Input parameter 1      | adc_x: indicates the selected ADC<br>This parameter can be ADC1.                                                                                                 |
| Input parameter 2      | adc_preempt_trig: indicates the selected trigger event for preempted group<br>This parameter can be any enumerated value in the<br>adc_preempt_trig_select_type. |
| Input parameter3       | new_state: the preset status of trigger mode<br>This parameter can be TRUE or FALSE.                                                                             |
| Output parameter       | NA                                                                                                                                                               |
| Return value           | NA                                                                                                                                                               |
| Required preconditions | NA                                                                                                                                                               |
| Called functions       | NA                                                                                                                                                               |

**adc\_preempt\_trig**

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

|                                |                   |
|--------------------------------|-------------------|
| ADC12_PREEMPT_TRIG_TMR1CH2:    | TMR1 CH2 event    |
| ADC12_PREEMPT_TRIG_TMR1CH3:    | TMR1 CH3 event    |
| ADC12_PREEMPT_TRIG_TMR2CH4:    | TMR2 CH4 event    |
| ADC12_PREEMPT_TRIG_TMR3CH4:    | TMR3 CH4 event    |
| ADC12_PREEMPT_TRIG_TMR15CH1:   | TMR15 CH1 event   |
| ADC12_PREEMPT_TRIG_TMR6TRGOUT: | TMR6 TRGOUT event |

ADC12\_PREEMPT\_TRIG\_EXINT15:  
ADC12\_PREEMPT\_TRIG\_SOFTWARE:

EXINT15 event  
Software trigger event

**Example:**

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

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

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

**Table 41. 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,<br>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<br>This parameter can be ADC1.                                                                             |
| Input parameter 2      | adc_preempt_channel: indicates the selected channel<br>Refer to <a href="#">adc_preempt_channel</a> for details.               |
| Input parameter3       | adc_offset_value: set the offset value for the selected channel<br>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

The 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);
```

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

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

**Table 42. 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<br>This parameter can be ADC1.                                                                |
| Input parameter 2      | adc_channel_count: indicates the number of ordinary group in partition mode<br>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.*

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

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

**Table 43. 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<br>This parameter can be ADC1.                                                               |
| Input parameter 2      | new_state: indicates the pre-configured status for partition mode of ordinary channels<br>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);
```

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

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

**Table 44. adc\_preempt\_part\_mode\_enable function**

| 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<br>This parameter can be ADC1.                                                                |
| Input parameter 2      | new_state: indicates the pre-configured status for partition mode of preempted channels<br>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);
```

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

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

**Table 45. 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<br>This parameter can be ADC1.                                                           |
| Input parameter 2      | new_state: indicates the pre-configured status for auto preempted group conversion<br>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);
```

## 5.2.24 adc\_tempersensor\_vinrv\_enable function

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

**Table 46. 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: the preset statue of internal temperature sensor and V <sub>INTRV</sub><br>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);
```

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

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

**Table 47. 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<br>This parameter can be ADC1.                                                                        |
| Input parameter 2      | new_state: indicates the pre-configured status for software-triggered ordinary group conversion<br>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);
```

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

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

**Table 48. 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<br>This parameter can be ADC1.                                                         |
| Output parameter       | NA                                                                                                                       |
| Return value           | flag_status: indicates the status of software-triggered ordinary group conversion<br>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));
```

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

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

**Table 49. 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                                                                                        |
| Input parameter 1      | adc_x: indicates the selected ADC<br>This parameter can be ADC1.                                                                        |
| Input parameter 2      | new_state: indicates the pre-configured status of software-triggered preempted group conversion<br>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);
```

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

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

**Table 50. 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<br>This parameter can be ADC1.                                                          |
| Output parameter       | NA                                                                                                                        |
| Return value           | flag_status: indicates the status of software-triggered preempted group conversion<br>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));
```

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

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

**Table 51. 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<br>This parameter can be ADC1.  |
| 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 a single ADC channel is configured.*

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

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

**Table 52. 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,<br>adc_preempt_channel_type adc_preempt_channel)       |
| Function description   | Get the converted data of preempted group                                                                        |
| Input parameter 1      | adc_x: indicates the selected ADC<br>This parameter can be ADC1.                                                 |
| Input parameter 2      | adc_preempt_channel: the selected preempted channel<br>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);
```

## 5.2.31 adc\_flag\_get function

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

**Table 53. 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<br>This parameter can be ADC1.                          |
| Input parameter 2      | adc_flag: indicates the selected flag<br>Refer to <a href="#">adc_flag</a> for details.   |
| Output parameter       | NA                                                                                        |
| Return value           | flag_status: the status for the selected flag bit.<br>This parameter 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 channel conversion
- ADC\_PCCS\_FLAG: Start of preempted channel conversion

ADC\_OCCS\_FLAG: Start of ordinary channel conversion

**Example:**

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

### 5.2.32 adc\_interrupt\_flag\_get function

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

**Table 54. 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 ADC interrupt flag status                                                               |
| Input parameter 1      | adc_x: indicates the selected ADC<br>This parameter can be ADC1.                            |
| Input parameter 2      | adc_flag: select a flag to be clear<br>Refer to <a href="#">adc_flag</a> below for details. |
| Output parameter       | NA                                                                                          |
| Return value           | flag_status: 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 channel conversion

**Example:**

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

## 5.2.33 adc\_flag\_clear function

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

**Table 55. 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<br>This parameter can be ADC1.           |
| Input parameter 2      | adc_flag: select a flag to be clear<br>Refer to <a href="#">adc_flag</a> . |
| 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);
```

## 5.3 Controller area network (CAN)

CAN register structure can\_type is defined in the “at32f425\_can.h”:

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

The table below gives a list of the CAN registers.

**Table 56. Summary of CAN 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 57. 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 disalbe/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          | Get the selected CAN interrupt flag status                                           |
| can_flag_clear                  | Clear the selected CAN flag                                                          |

### 5.3.1 can\_reset function

The table below describes the function can\_reset.

**Table 58. 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      | can_x: indicates the selected CAN<br>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 59. 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 60. 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      | can_x: indicates the selected CAN<br>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                                                                                      |

The can\_baudrate\_type is defined in the at32f425\_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 1 has 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 61. 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 62. 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      | can_x: the selected CAN peripheral<br>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 the can_base_type before starting.    |
| Called functions       | NA                                                                            |

The can\_base\_type is defined in the at32f425\_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 mod               |
| CAN_MODE_LISTENONLY:          | Listen only mode           |
| CAN_MODE_LISTENONLY_LOOPBACK: | Loopback + listen only mod |

**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 number 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.prsf_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 63. 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 64. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.                     |
| 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 necessary to define a variable of can_filter_init_type before starting.        |
| Called functions       | NA                                                                                   |

The can\_filter\_init\_type is defined in the at32f425\_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 65. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.                |
| Input parameter 2      | new_state: Enable or disable<br>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 66. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.     |
| Input parameter 2      | new_state: Enable or disable<br>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 67. 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      | can_x: indicates the selected CAN<br>This parameter can be 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                                                                                                |

The can\_tx\_message\_type is defined in the at32f425\_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: 0x000~0x1FFFFFFF

**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

**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 68. 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      | can_x: indicates the selected CAN<br>This parameter can be 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 69. 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      | can_x: indicates the selected CAN<br>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 70. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.                                                     |
| Input parameter 2      | fifo_number: receive FIFO<br>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);                                    |

The can\_rx\_message\_type is defined in the at32f425\_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 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

**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 71. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.                        |
| Input parameter 2      | fifo_number: receive FIFO number<br>This parameter can be 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 72. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.                            |
| Input parameter 2      | fifo_number: receive FIFO number<br>This parameter can be 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 73. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.                                  |
| Input parameter 2      | <i>can_operating_mode</i> : 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 74. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.           |
| Output parameter       | NA                                                                         |
| Return value           | <i>can_enter_doze_status</i> : 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 75. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.                      |
| Output parameter       | NA                                                                                    |
| Return value           | <a href="#">can_quit_doze_status</a> : 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 76. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.  |
| Output parameter       | NA                                                                |
| Return value           | <a href="#">can_error_record</a> : 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 77. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.       |
| Output parameter       | NA                                                                     |
| Return value           | receive_error_counter: Receive error counter<br>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 78. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.         |
| Output parameter       | NA                                                                       |
| Return value           | transmit_error_counter: Transmit error counter<br>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 79. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.                       |
| Input parameter 2      | <a href="#">can_int</a> : Select CAN interrupts                                        |
| Input parameter3       | new_state: Enable or disable<br>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 80. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1.                                       |
| Input parameter 2      | <i>can_flag</i> : indicates the selected flag<br>Refer to the “can_flag” description below for details |
| Output parameter       | NA                                                                                                     |
| Return value           | flag_status: the status of the selected flag<br>Return value can be SET or RESET.                      |
| 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\_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\_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: Quit 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 81. 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 status                                           |
| Input parameter 1      | can_x: indicates the selected CAN<br>This parameter can be CAN1, CNA2   |
| Input parameter 2      | <i>can_flag</i> : indicates the selected flag Refer to can_flag         |
| Output parameter       | NA                                                                      |
| Return value           | flag_status: SET or RESET                                               |
| 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\_RF0MN\_FLAG: FIFO0 not-empty flag
- CAN\_RF0FF\_FLAG: FIFO0 full flag
- CAN\_RF0OF\_FLAG: FIFO0 overflow flag
- CAN\_RF1MN\_FLAG: FIFO1 not-empty flag
- CAN\_RF1FF\_FLAG: FIFO1 full flag
- CAN\_RF1OF\_FLAG: FIFO1 overflow flag
- CAN\_QDZIF\_FLAG: Quit 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 82. 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      | can_x: indicates the selected CAN<br>This parameter can be CAN1. |
| Input parameter 2      | <i>can_flag</i> : indicates the selected flag Refer to can_flag  |
| 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\_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\_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: Quit 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 “`at32f425_crc.h`”:

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

The table below gives a list of the CRC registers.

**Table 83. Summary of CRC registers**

| Register | Description                   |
|----------|-------------------------------|
| dt       | Data register                 |
| cdt      | General-purpose data register |
| ctrl     | Control register              |
| idt      | Control register              |
| poly     | Polynomial generator          |

The table below gives a list of CRC library functions.

**Table 84. Summary of CRC library functions**

| Function name                            | Description                                                                               |
|------------------------------------------|-------------------------------------------------------------------------------------------|
| <code>crc_data_reset</code>              | Data register reset                                                                       |
| <code>crc_one_word_calculate</code>      | Calculate 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_date_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 valid width                                                                |
| <code>crc_poly_size_get</code>           | Get polynomial valid width                                                                |

## 5.4.1 crc\_data\_reset function

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

**Table 85. 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 86. 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 a 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 87. 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 88. 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 89. 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\_data\_get function

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

**Table 90. crc\_common\_data\_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_get();
```

### 5.4.7 crc\_init\_data\_set function

The table below describes the function `crc_init_data_set`.

**Table 91. `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 92. `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. Refer to “value” below for details.     |
| Input parameter 2      | NA                                                                          |
| Output parameter       | NA                                                                          |
| Return value           | NA                                                                          |
| Required preconditions | NA                                                                          |
| Called functions       | NA                                                                          |

**value**

Define 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 93. crc\_reverse\_output\_data\_set function**

| Name                   | Description                                                              |
|------------------------|--------------------------------------------------------------------------|
| Function name          | crc_reverse_output_data_set                                              |
| Function prototype     | void crc_reverse_output_data_set(crc_reverse_output_type value);         |
| Function description   | Define the CRC output data reverse type                                  |
| Input parameter 1      | value: output data bit reverse type. Refer to “value” below for details. |
| Input parameter 2      | NA                                                                       |
| Output parameter       | NA                                                                       |
| Return value           | NA                                                                       |
| Required preconditions | NA                                                                       |
| Called functions       | NA                                                                       |

### value

Define 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 94. crc\_poly\_value\_set function**

| Name                   | Description                              |
|------------------------|------------------------------------------|
| Function name          | crc_poly_value_set                       |
| Function prototype     | void crc_poly_value_set(uint32_t value); |
| 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 95. `crc_poly_value_get` function**

| Name                   | Description                                     |
|------------------------|-------------------------------------------------|
| Function name          | <code>crc_poly_value_get</code>                 |
| Function prototype     | <code>uint32_t crc_poly_value_get(void);</code> |
| Function description   | Get CRC polynomial value                        |
| Input parameter 1      | NA                                              |
| Input parameter 2      | NA                                              |
| Output parameter       | NA                                              |
| Return value           | <code>uint32_t</code> : return 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 96. `crc_poly_size_set` function**

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

**size**

Define the valid width of polynomial.

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

**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 97. 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 valid width              |
| Input parameter 1      | NA                                          |
| Input parameter 2      | NA                                          |
| Output parameter       | NA                                          |
| Return value           | crc_poly_size_type: polynomial valid width  |
| Required preconditions | NA                                          |
| Called functions       | NA                                          |

### crc\_poly\_size\_type

Define the valid width of polynomial.

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

### 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 “`at32f425_crm.h`”:

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

The table below gives a list of the CRM registers.

**Table 98. 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 enable register   |
| apb1en      | APB1 peripheral clock enable register   |
| bpdः        | Battery powered domain control register |
| ctrlsts     | Control/status register                 |
| ahbrst      | AHB peripheral reset register           |
| pll         | PLL configuration register              |
| misc1       | Extra register 1                        |
| otg_extctrl | OTG Extra control register              |
| misc2       | Extra register 2                        |

The table below gives a list of CRM library functions.

**Table 99. 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 higded 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 | Enable peripheral clock in sleep mode                                     |
| crm_clock_source_enable            | Clock source enable                                                       |
| crm_flag_clear                     | Clear flag                                                                |
| crm_ertc_clock_select              | ERTC clock source selection                                               |
| crm_ertc_clock_enable              | ERTC 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              | Set ADC clock division                                                    |
| crm_usb_clock_div_set              | USB clock division                                                        |
| crm_clock_failure_detection_enable | Clock failure detection enable                                            |
| crm_battery_powered_domain_reset   | Battery powered domain reset                                              |
| crm_pll_config                     | Configure PLL clock source and frequency multiplication factor            |
| crm_pll_config2                    | Configure PLL clock source and frequency multiplication factor 2          |
| 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                                                          |
| crm_hick_sclk_frequency_select     | Set system clock frequency as 8M or 48M when HICK is used as system clock |
| crm_usb_clock_source_select        | Select PLL or interal high-speed clock (48M) as USB clock source          |
| crm_clkout_div_set                 | CLKOUT output clock division                                              |
| crm_usbdv_reset                    | USB divider reset                                                         |
| crm_pll_parameter_calculate        | Calculate PLL parameters automatically                                    |

## 5.5.1 crm\_reset function

The table below describes the function crm\_reset.

**Table 100. 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] in 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 101. 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 102. 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 103. 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 104. 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   | Get CRM interrupt flag status                                 |
| Input parameter 1      | Flag: select a flag<br>Refer to the "flag" below for details. |
| Input parameter 2      | NA                                                            |
| Output parameter       | NA                                                            |
| Return value           | flag_status: 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 105. 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 106. 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 0x20, configurable range is from 0 to 0x3F. |
| 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 107. 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

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

**Table 108. 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 at32f425\_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

### 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 109. 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                                                 |
| 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 at32f425\_crm.h.

The naming rule of this parameter is: CRM\_peripheral\_PERIPH\_RESET.

CRM\_DMA1\_PERIPH\_RESET: DMA1 peripheral reset

CRM\_DMA2\_PERIPH\_RESET: DMA2 peripheral reset

...

CRM\_PWC\_PERIPH\_RESET: PWC 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 110. `crm_periph_sleep_mode_clock_enable` function**

| Name                   | Description                                                                                                         |
|------------------------|---------------------------------------------------------------------------------------------------------------------|
| Function name          | <code>crm_periph_sleep_mode_clock_enable</code>                                                                     |
| Function prototype     | <code>void crm_periph_sleep_mode_clock_enable(crm_periph_clock_sleepmd_type value, confirm_state new_state);</code> |
| Function description   | Peripheral clock enable in Sleep mode                                                                               |
| Input parameter 1      | Value: peripheral clock type in Sleep mode                                                                          |
| Input parameter 2      | <code>new_state</code> : TRUE or FALSE                                                                              |
| Output parameter       | NA                                                                                                                  |
| Return value           | NA                                                                                                                  |
| Required preconditions | NA                                                                                                                  |
| Called functions       | NA                                                                                                                  |

**value**

It indicates the selected peripheral. `crm_periph_clock_sleepmd_type` is defined in `at32f425_crm.h`.

The naming rule of this parameter is: CRM\_peripheral name \_ PERIPH\_CLOCK\_SLEEP\_MODE.

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

CRM\_FLASH\_PERIPH\_RESET: FLASH clock definition 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 function.

**Table 111. 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 function.

**Table 112. 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   |
| 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\_ertc\_clock\_select function

The table below describes the function crm\_ertc\_clock\_select function.

**Table 113. crm\_ertc\_clock\_select function**

| Name                   | Description                                            |
|------------------------|--------------------------------------------------------|
| Function name          | crm_ertc_clock_select                                  |
| Function prototype     | void crm_ertc_clock_select(crm_ertc_clock_type value); |
| Function description   | Select ERTC clock source                               |
| Input parameter 1      | value: indicates ertc 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_ERTC_CLOCK_NOCLK:    | No clock source for ERTC        |
| CRM_ERTC_CLOCK_LEXT:     | LEXT selected as ERTC clock     |
| CRM_ERTC_CLOCK_LICK:     | LICK selected as ERTC clock     |
| CRM_ERTC_CLOCK_HEXT_DIV: | HEXT/128 selected as ERTC clock |

**Example:**

```
/* config lext as ertc clock */
crm_ertc_clock_select (CRM_ERTC_CLOCK_LEXT);
```

## 5.5.15 crm\_ertc\_clock\_enable function

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

**Table 114. crm\_ertc\_clock\_enable function**

| Name                   | Description                                          |
|------------------------|------------------------------------------------------|
| Function name          | crm_ertc_clock_enable                                |
| Function prototype     | void crm_ertc_clock_enable(confirm_state new_state); |
| Function description   | Enable ERTC 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 ertc clock */
crm_ertc_clock_enable (TRUE);
```

## 5.5.16 crm\_ahb\_div\_set function

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

**Table 115. 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          |
| 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 116. 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 117. 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 118. 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: 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 as ADC clock
- CRM\_ADC\_DIV\_4: APB/4 as ADC clock
- CRM\_ADC\_DIV\_6: APB/6 as ADC clock
- CRM\_ADC\_DIV\_8: APB/8 as ADC clock
- CRM\_ADC\_DIV\_12: APB/12 as ADC clock
- CRM\_ADC\_DIV\_16: APB/16 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 119. 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 PLL clock division                            |
| Input parameter 1      | div_value: division factor                              |
| 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 cock
- CRM\_USB\_DIV\_1: PLL/1 as USB cock
- CRM\_USB\_DIV\_2\_5: PLL/2.5 as USB cock
- CRM\_USB\_DIV\_2: PLL/2 as USB cock
- CRM\_USB\_DIV\_3\_5: PLL/3.5 as USB cock

CRM\_USB\_DIV\_3: PLL/3 as USB cock  
 CRM\_USB\_DIV\_4: PLL/4 as USB cock

**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 120. `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 121. `crm_batteryPoweredDomainReset`**

| 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 122. 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); |
| Function description   | Configure PLL clock source and frequency multiplication factor                             |
| Input parameter 1      | clock_source: clock source for PLL frequency multiplication                                |
| Input parameter 2      | mult_value: multiplication factor                                                          |
| Output parameter       | NA                                                                                         |
| Return value           | NA                                                                                         |
| Required preconditions | PLL clock source must be enabled and stabilized before configuring and enabling PLL        |
| Called functions       | NA                                                                                         |

### clock\_source

CRM\_PLL\_SOURCE\_HICK: HICK is selected as PLL clock source

CRM\_PLL\_SOURCE\_HEXT: HEXT is selected as PLL clock source

CRM\_PLL\_SOURCE\_HEXT\_DIV: divided HEXT is selected as PLL clock source

### mult\_value

CRM\_PLL\_MULT\_2: 2x input clock

CRM\_PLL\_MULT\_3: 3x input clock

...

CRM\_PLL\_MULT\_63: 63 x input clock

CRM\_PLL\_MULT\_64: 64x input clock

### Example:

```
/* config pll clock resource */
crm_pll_config(CRM_PLL_SOURCE_HEXT_DIV, CRM_PLL_MULT_30);
```

## 5.5.24 crm\_pll\_config2 function

The table below describes the function crm\_pll\_config2.

**Table 123. crm\_pll\_config2 function**

| Name                   | Description                                                                                                             |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| Function name          | crm_pll_config2                                                                                                         |
| Function prototype     | void crm_pll_config2(crm_pll_clock_source_type clock_source, uint16_t pll_ns, uint16_t pll_ms, crm_pll_fr_type pll_fr); |
| Function description   | Configure PLL clock source and frequency multiplication and division factor                                             |
| Input parameter 1      | clock_source: clock source for PLL frequency multiplication                                                             |
| Input parameter 2      | pll_ns: frequency multiplication factor from 31 to 500                                                                  |
| Input parameter3       | pll_ms: pre-division frequency factor from 1 to 15                                                                      |
| Input parameter4       | pll_fr: post-division frequency factor                                                                                  |
| Output parameter       | NA                                                                                                                      |
| Return value           | NA                                                                                                                      |
| Required preconditions | PLL clock source must be enabled and stabilized before configuring and enabling PLL                                     |
| Called functions       | NA                                                                                                                      |

Frequency multiplication formula:  $\text{PLLCLK} = \text{PLL input clock} / \text{PLL_MS} * \text{PLL_NS} / \text{PLL_FR}$

Requirements:

$2\text{MHz} \leq \text{PLL input clock} / \text{PLL_MS} \leq 16\text{MHz}$

$500\text{MHz} \leq \text{PLL input clock} / \text{PLL_MS} * \text{PLL_NS} \leq 1000\text{MHz}$

**clock\_source**

CRM\_PLL\_SOURCE\_HICK: HICK is selected as PLL clock source

CRM\_PLL\_SOURCE\_HEXT: HEXT is selected as PLL clock source

CRM\_PLL\_SOURCE\_HEXT\_DIV: divided HEXT is selected as PLL clock source

**pll\_fr**

CRM\_PLL\_FR\_1: PLL/1

CRM\_PLL\_FR\_2: PLL/2

CRM\_PLL\_FR\_4: PLL/4

CRM\_PLL\_FR\_8: PLL/8

CRM\_PLL\_FR\_16: PLL/16

CRM\_PLL\_FR\_32: PLL/32

**Example:**

```
/* config pll clock resource */
crm_pll_config2(CRM_PLL_SOURCE_HEXT_DIV, 96, 1, CRM_PLL_FR_8);
```

## 5.5.25 crm\_sysclk\_switch function

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

**Table 124. 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 |
| 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);
```

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

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

**Table 125. 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)
{
}
```

## 5.5.27 crm\_clocks\_freq\_get function

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

**Table 126. 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 |
| 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 at32f425\_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

Return ADC clock frequency, in Hz

### Example:

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

## 5.5.28 crm\_clock\_out\_set function

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

**Table 127. 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   | Select clock source output on clkout pin               |
| Input parameter 1      | clkout: clock source output on 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);
```

## 5.5.29 crm\_interrupt\_enable function

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

**Table 128. 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                             |
| 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);
```

### 5.5.30 crm\_auto\_step\_mode\_enable function

The table below describes the function crm\_auto\_step\_mode\_enable.

**Table 129. crm\_auto\_step\_mode\_enable function**

| Name                   | Description                                              |
|------------------------|----------------------------------------------------------|
| Function name          | crm_auto_step_mode_enable                                |
| Function prototype     | void crm_auto_step_mode_enable(confirm_state new_state); |
| Function description   | Enable auto step-by-step mode                            |
| Input parameter 1      | new_state: Enable (TRUE), disable (FALSE)                |
| Input parameter 2      | NA                                                       |
| Output parameter       | NA                                                       |
| Return value           | NA                                                       |
| Required preconditions | NA                                                       |
| Called functions       | NA                                                       |

**Example:**

```
/* enable auto step mode */
crm_auto_step_mode_enable(TRUE);
```

### 5.5.31 crm\_hick\_sclk\_frequency\_select function

The table below describes the function crm\_hick\_sclk\_frequency\_select.

**Table 130. crm\_hick\_sclk\_frequency\_select function**

| Name                   | Description                                                               |
|------------------------|---------------------------------------------------------------------------|
| Function name          | crm_hick_sclk_frequency_select                                            |
| Function prototype     | void crm_hick_sclk_frequency_select(crm_hick_sclk_frequency_type value);  |
| Function description   | Select 8M or 48M system clock frequency when HICK is used as system clock |
| Input parameter 1      | value: 8M or 48M HICK                                                     |
| Input parameter 2      | NA                                                                        |
| Output parameter       | NA                                                                        |
| Return value           | NA                                                                        |
| Required preconditions | NA                                                                        |
| Called functions       | NA                                                                        |

**value**

CRM\_HICK\_SCLK\_8MHZ: 8MHz HICK used as system clock

CRM\_HICK\_SCLK\_48MHZ: 48MHz HICK used as system clock

**Example:**

```
/* config sysclk with hick 48mhz */
crm_hick_sclk_frequency_select(CRM_HICK_SCLK_48MHZ);
```

### 5.5.32 crm\_usb\_clock\_source\_select function

The table below describes the function crm\_usb\_clock\_source\_select.

**Table 131. crm\_usb\_clock\_source\_select function**

| Name                   | Description                                                        |
|------------------------|--------------------------------------------------------------------|
| Function name          | crm_usb_clock_source_select                                        |
| Function prototype     | void crm_usb_clock_source_select(crm_usb_clock_source_type value); |
| Function description   | Select PLL or internal high-speed clock (48M) as USB clock source  |
| Input parameter 1      | value: PLL or internal high-speed clock (48M)                      |
| Input parameter 2      | NA                                                                 |
| Output parameter       | NA                                                                 |
| Return value           | NA                                                                 |
| Required preconditions | NA                                                                 |
| Called functions       | NA                                                                 |

**value**

CRM\_USB\_CLOCK\_SOURCE\_PLL: PLL is used as USB clock source

CRM\_USB\_CLOCK\_SOURCE\_HICK: HICK is used as USB clock source

**Example:**

```
/* select hick48 as usb clock */
crm_usb_clock_source_select (CRM_USB_CLOCK_SOURCE_HICK);
```

### 5.5.33 crm\_clkout\_div\_set function

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

**Table 132. crm\_clkout\_div\_set function**

| Name                   | Description                                              |
|------------------------|----------------------------------------------------------|
| Function name          | crm_clkout_div_set                                       |
| Function prototype     | void crm_clkout_div_set(crm_clkout_div_type clkout_div); |
| Function description   | Set clkout output clock division                         |
| Input parameter 1      | clkout_div: clkout output frequency division value       |
| Input parameter 2      | NA                                                       |
| Output parameter       | NA                                                       |
| Return value           | NA                                                       |
| Required preconditions | NA                                                       |
| Called functions       | NA                                                       |

**value**

CRM\_CLKOUT\_DIV\_1: internal clock/1 output

CRM\_CLKOUT\_DIV\_2: internal clock/2 output

CRM\_CLKOUT\_DIV\_4: internal clock/4 output

CRM\_CLKOUT\_DIV\_8: internal clock/8 output

CRM\_CLKOUT\_DIV\_16: internal clock/16 output

CRM\_CLKOUT\_DIV\_64: internal clock/64 output

CRM\_CLKOUT\_DIV\_128: internal clock/128 output

CRM\_CLKOUT\_DIV\_256: internal clock/256 output

CRM\_CLKOUT\_DIV\_512: internal clock/512 output

**Example:**

```
/* config clkout division */
crm_clkout_div_set(CRM_CLKOUT_DIV_1);
```

### 5.5.34 crm\_usbdv\_reset function

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

**Table 133. crm\_usbdv\_reset function**

| Name                   | Description                 |
|------------------------|-----------------------------|
| Function name          | crm_usbdv_reset             |
| Function prototype     | void crm_usbdv_reset(void); |
| Function description   | USB divider reset           |
| Input parameter 1      | NA                          |
| Input parameter 2      | NA                          |
| Output parameter       | NA                          |
| Return value           | NA                          |
| Required preconditions | NA                          |
| Called functions       | NA                          |

**Example:**

```
/* reset usbdv */
crm_usbdv_reset();
```

### 5.5.35 crm\_pll\_parameter\_calculate function

The table below describes the function crm\_pll\_parameter\_calculate.

**Table 134. crm\_pll\_parameter\_calculate function**

| Name                   | Description                                                                                                                                                                          |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | crm_pll_parameter_calculate                                                                                                                                                          |
| Function prototype     | error_status crm_pll_parameter_calculate(crm_pll_clock_source_type pll_rcs,<br>uint32_t target_sclk_freq, uint16_t *ret_ms, uint16_t *ret_ns, uint16_t *ret_fr);                     |
| Function description   | PLL parameter auto calculation                                                                                                                                                       |
| Input parameter 1      | pll_rcs: pll input clock source                                                                                                                                                      |
| Input parameter 2      | target_sclk_freq: target clock frequency multiplication, for example, for 200 MHz,<br>this parameter can be target_sclk_freq=200000000.                                              |
| Output parameter1      | ret_ms: return pll_ms parameter                                                                                                                                                      |
| Output parameter2      | ret_ns: return pll_ns parameter                                                                                                                                                      |
| Output parameter3      | ret_fr: return pll_fr parameter                                                                                                                                                      |
| Return value           | error_status: Calculation status.<br>SUCCESS: the calculated result equals the target clock PLL parameter<br>ERROR: the calculated result is close to the target clock PLL parameter |
| Required preconditions | NA                                                                                                                                                                                   |
| Called functions       | NA                                                                                                                                                                                   |

**Example:**

```
/* pll parameter calculate automatic */
uint16_t pll_ms = 0, pll_ns = 0, pll_fr = 0;
crm_pll_parameter_calculate (CRM_PLL_SOURCE_HEXT, 200000000, &pll_ms, &pll_ns, &pll_fr);
```

## 5.6 Debug

The DEBUG register structure debug\_type is defined in the “at32f425\_debug.h”:

```
/*
 * @brief type define debug register all
 */
typedef struct
{
 ...
} debug_type;
```

The table below gives a list of the DEBUG registers.

**Table 135. Summary of DEBUG registers**

| Register | Description      |
|----------|------------------|
| idcode   | Device ID        |
| ctrl     | Control register |

The table below gives a list of DEBUG library functions.

**Table 136. Summary of DEBUG library functions**

| Function name         | Description                             |
|-----------------------|-----------------------------------------|
| debug_device_id_get   | Read device idcode                      |
| debug_periph_mode_set | Set corresponding peripheral debug mode |

### 5.6.1 debug\_device\_id\_get function

The table below describes the function debug\_device\_id\_get.

**Table 137. 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 idcode                  |
| Input parameter 1      | NA                                  |
| Input parameter 2      | NA                                  |
| Output parameter       | NA                                  |
| Return value           | Return 32-bit idcode                |
| Required preconditions | NA                                  |
| Called functions       | NA                                  |

**Example:**

```
/* get idcode */
uint32_t idcode = 0;
idcode = debug_device_id_get();
```

## 5.6.2 debug\_periph\_mode\_set function

The table below describes the function debug\_periph\_mode\_set.

**Table 138. 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   | Debug settings for the selected peripheral or mode                               |
| Input parameter 1      | periph_debug_mode: Selected 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

Debug settings for the selected peripheral or mode.

|                           |                                             |
|---------------------------|---------------------------------------------|
| DEBUG_SLEEP:              | DEBUG settings in SLEEP mode                |
| DEBUG_DEEPSLEEP:          | DEBUG settings in DEEPSLEEP mode            |
| DEBUG_STANDBY:            | DEBUG settings in STANDBY mode              |
| DEBUG_WDT_PAUSE:          | DEBUG settings for WDT pause                |
| DEBUG_WWDT_PAUSE:         | DEBUG settings for WWDT pause               |
| DEBUG_TMR1_PAUSE:         | DEBUG settings for TMR1 pause               |
| DEBUG_TMR2_PAUSE:         | DEBUG settings for TMR2 pause               |
| DEBUG_TMR3_PAUSE:         | DEBUG settings for TMR3 pause               |
| DEBUG_TMR6_PAUSE:         | DEBUG settings for TMR6 pause               |
| DEBUG_TMR7_PAUSE:         | DEBUG settings for TMR7 pause               |
| DEBUG_TMR13_PAUSE:        | DEBUG settings for TMR13 pause              |
| DEBUG_TMR14_PAUSE:        | DEBUG settings for TMR14 pause              |
| DEBUG_TMR15_PAUSE:        | DEBUG settings for TMR15 pause              |
| DEBUG_TMR16_PAUSE:        | DEBUG settings for TMR16 pause              |
| DEBUG_TMR17_PAUSE:        | DEBUG settings for TMR17 pause              |
| DEBUG_I2C1_SMBUS_TIMEOUT: | DEBUG settings for I2C1 SMBUS TIMEOUT pause |
| DEBUG_I2C2_SMBUS_TIMEOUT: | DEBUG settings for I2C2 SMBUS TIMEOUT pause |
| DEBUG_CAN1_PAUSE:         | DEBUG settings for CAN1 pause               |
| DEBUG_ERTC_PAUSE:         | DEBUG settings for ERTC pause               |
| DEBUG_ERTC_512_PAUSE:     | DEBUG settings for ERTC 512Hz pause         |

### Example:

```
/* enable tmr1 debug mode */
debug_periph_mode_set(DEBUG_TMR1_PAUSE, TRUE);
```

## 5.7 DMA controller

The DMA register structure `dma_type` is defined in the “`at32f425_dma.h`”:

```
/*
 * @brief type define dma register
 */
typedef struct
{
 ...
}
```

`} dma_type;`

DMA channel register structure `dma_channel_type` is defined in the “`at32f425_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 139. 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     |
| dma_src_sel0 | Channel source register 0                 |
| dma_src_sel1 | Channel source register 1                 |

The table below gives a list of DMA library functions.

**Table 140. 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 DMA 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                     |

### 5.7.1 **dma\_default\_para\_init** function

The table below describes the function `dma_default_para_init`.

**Table 141. `dma_default_para_init` function**

| Name                   | Description                                                              |
|------------------------|--------------------------------------------------------------------------|
| Function name          | <code>dma_default_para_init</code>                                       |
| Function prototype     | <code>void dma_default_para_init(dma_init_type* dma_init_struct);</code> |
| Function description   | Initialize the parameters of the <code>dma_init_struct</code>            |
| Input parameter 1      | <code>dma_init_struct</code> : <code>dma_init_type</code> 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 142. `dma_init_struct` default values**

| Member                             | Default values                              |
|------------------------------------|---------------------------------------------|
| <code>peripheral_base_addr</code>  | 0x0                                         |
| <code>memory_base_addr</code>      | 0x0                                         |
| <code>direction</code>             | <code>DMA_DIR_PERIPHERAL_TO_MEMORY</code>   |
| <code>buffer_size</code>           | 0x0                                         |
| <code>peripheral_inc_enable</code> | FALSE                                       |
| <code>memory_inc_enable</code>     | FALSE                                       |
| <code>peripheral_data_width</code> | <code>DMA_PERIPHERAL_DATA_WIDTH_BYTE</code> |
| <code>memory_data_width</code>     | <code>DMA_MEMORY_DATA_WIDTH_BYTE</code>     |
| <code>loop_mode_enable</code>      | FALSE                                       |
| <code>priority</code>              | <code>DMA_PRIORITY_LOW</code>               |

Example:

```
/* dma init config with its default value */
dma_init_type dma_init_struct = {0};
dma_default_para_init(&dma_init_struct);
```

## 5.7.2 `dma_init` function

The table below describes the function `dma_init`.

**Table 143. `dma_init` function**

| Name                   | Description                                                                                |
|------------------------|--------------------------------------------------------------------------------------------|
| Function name          | <code>dma_init</code>                                                                      |
| Function prototype     | <code>void dma_init(dma_channel_type* dmax_channel, dma_init_type* dma_init_struct)</code> |
| Function description   | Initialize 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>dma_init_struct</code> : <code>dma_init_type</code> 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 `at32f425_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(DMA1_CHANNEL1, &dma_init_struct);

```

### 5.7.3 dma\_reset function

The table below describes the function dma\_reset.

**Table 144. 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, y=1...7 |
| Output parameter       | NA                                                                     |
| Return value           | NA                                                                     |
| Required preconditions | NA                                                                     |
| Called functions       | NA                                                                     |

**Example:**

```

/* reset dma1 channel1 */
dma_reset(DMA1_CHANNEL1);

```

### 5.7.4 dma\_data\_number\_set function

The table below describes the function dma\_data\_number\_set.

**Table 145. 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, 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 dma1 channel1 data count is 0x100*/
dma_data_number_set(DMA1_CHANNEL1, 0x100);

```

## 5.7.5 dma\_data\_number\_get function

The table below describes the function `dma_data_number_get`.

**Table 146. `dma_data_number_get` function**

| Name                   | Description                                                                 |
|------------------------|-----------------------------------------------------------------------------|
| Function name          | <code>dma_data_number_get</code>                                            |
| Function prototype     | <code>uint16_t dma_data_number_get(dma_channel_type* dmax_channely);</code> |
| 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,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 dma1 channel1 data count*/
uint16_t data_counter;
data_counter = dma_data_number_get(DMA1_CHANNEL1);
```

## 5.7.6 dma\_interrupt\_enable function

The table below describes the function `dma_interrupt_enable`.

**Table 147. `dma_interrupt_enable` function**

| Name                   | Description                                                                                                         |
|------------------------|---------------------------------------------------------------------------------------------------------------------|
| Function name          | <code>dma_interrupt_enable</code>                                                                                   |
| Function prototype     | <code>void dma_interrupt_enable(dma_channel_type* dmax_channely, uint32_t dma_int, confirm_state new_state);</code> |
| Function description   | Enable DMA channels interrupt                                                                                       |
| Input parameter 1      | dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1, y=1...7                                             |
| Input parameter 2      | dma_int: interrupt source selection                                                                                 |
| Input parameter3       | 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 dma1 channel1 transfer full data interrupt */
dma_interrupt_enable(DMA1_CHANNEL1, DMA_FDT_INT, TRUE);
```

## 5.7.7 dma\_channel\_enable function

The table below describes the function dma\_channel\_enable.

**Table 148. dma\_channel\_enable function**

| Name                   | Description                                                                        |
|------------------------|------------------------------------------------------------------------------------|
| Function name          | dma_channel_enable                                                                 |
| Function prototype     | void dma_channel_enable(dma_channel_type* dmax_channely, confirm_state new_state); |
| Function description   | Enable the selected DMA channel                                                    |
| Input parameter 1      | dmax_channely: DMAx_CHANNELy defines a DMA channel number, x=1, y=1...7            |
| Input parameter 2      | new_state: 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: Disabled

TRUE: Enabled

### **Example:**

```
/* enable dma channel */
dma_channel_enable(DMA1_CHANNEL1, TRUE);
```

## 5.7.8 dma\_flexible\_config function

The table below describes the function dma\_flexible\_enable.

**Table 149. dma\_flexible\_config function**

| Name                   | Description                                                                                                   |
|------------------------|---------------------------------------------------------------------------------------------------------------|
| Function name          | dma_flexible_config                                                                                           |
| Function prototype     | void dma_flexible_config(dma_type* dma_x, uint8_t flex_channelx, dma_flexible_request_type flexible_request); |
| Function description   | Configure flexible request mapping                                                                            |
| Input parameter 1      | dma_x: DMAx, x=1                                                                                              |
| Input parameter 2      | flex_channelx: FLEX_CHANNELx channel number, x=1...7                                                          |
| Input parameter 3      | flexible_request: flexible mapping request source ID                                                          |
| Output parameter       | NA                                                                                                            |
| Return value           | NA                                                                                                            |
| Required preconditions | NA                                                                                                            |
| Called functions       | NA                                                                                                            |

### **flexible\_request**

The table below shows flexible mapping request source ID:

Table 150. Flexible mapping request source ID

| Request source ID | Description                 | Request source ID | Description                 |
|-------------------|-----------------------------|-------------------|-----------------------------|
| 0x05              | DMA_FLEXIBLE_ADC1           | 0x10              | DMA_FLEXIBLE_SPI1_RX        |
| 0x11              | DMA_FLEXIBLE_SPI1_TX        | 0x12              | DMA_FLEXIBLE_SPI2_RX        |
| 0x13              | DMA_FLEXIBLE_SPI2_TX        | 0x3C              | DMA_FLEXIBLE_SPI3_RX        |
| 0x3D              | DMA_FLEXIBLE_SPI3_TX        | 0x32              | DMA_FLEXIBLE_UART1_RX       |
| 0x33              | DMA_FLEXIBLE_UART1_TX       | 0x34              | DMA_FLEXIBLE_UART2_RX       |
| 0x35              | DMA_FLEXIBLE_UART2_TX       | 0x36              | DMA_FLEXIBLE_UART3_RX       |
| 0x37              | DMA_FLEXIBLE_UART3_TX       | 0x38              | DMA_FLEXIBLE_UART4_RX       |
| 0x39              | DMA_FLEXIBLE_UART4_TX       | 0x0A              | DMA_FLEXIBLE_I2C1_RX        |
| 0x0B              | DMA_FLEXIBLE_I2C1_TX        | 0x0C              | DMA_FLEXIBLE_I2C2_RX        |
| 0x0D              | DMA_FLEXIBLE_I2C2_TX        | 0x18              | DMA_FLEXIBLE_TMR1_TRIG      |
| 0x18              | DMA_FLEXIBLE_TMR1_HALL      | 0x19              | DMA_FLEXIBLE_TMR1_OVERFLOW  |
| 0x14              | DMA_FLEXIBLE_TMR1_CH1       | 0x15              | DMA_FLEXIBLE_TMR1_CH2       |
| 0x16              | DMA_FLEXIBLE_TMR1_CH3       | 0x17              | DMA_FLEXIBLE_TMR1_CH4       |
| 0x1E              | DMA_FLEXIBLE_TMR2_TRIG      | 0x1F              | DMA_FLEXIBLE_TMR2_OVERFLOW  |
| 0x1A              | DMA_FLEXIBLE_TMR2_CH1       | 0x1B              | DMA_FLEXIBLE_TMR2_CH2       |
| 0x1C              | DMA_FLEXIBLE_TMR2_CH3       | 0x1D              | DMA_FLEXIBLE_TMR2_CH4       |
| 0x24              | DMA_FLEXIBLE_TMR3_TRIG      | 0x25              | DMA_FLEXIBLE_TMR3_OVERFLOW  |
| 0x20              | DMA_FLEXIBLE_TMR3_CH1       | 0x21              | DMA_FLEXIBLE_TMR3_CH2       |
| 0x22              | DMA_FLEXIBLE_TMR3_CH3       | 0x23              | DMA_FLEXIBLE_TMR3_CH4       |
| 0x26              | DMA_FLEXIBLE_TMR6_OVERFLOW  | 0x27              | DMA_FLEXIBLE_TMR7_OVERFLOW  |
| 0x2A              | DMA_FLEXIBLE_TMR15_TRIG     | 0x2A              | DMA_FLEXIBLE_TMR15_HALL     |
| 0x2B              | DMA_FLEXIBLE_TMR15_OVERFLOW | 0x28              | DMA_FLEXIBLE_TMR15_CH1      |
| 0x29              | DMA_FLEXIBLE_TMR15_CH2      | 0x2E              | DMA_FLEXIBLE_TMR16_OVERFLOW |
| 0x2C              | DMA_FLEXIBLE_TMR16_CH1      | 0x31              | DMA_FLEXIBLE_TMR17_OVERFLOW |
| 0x2F              | DMA_FLEXIBLE_TMR17_CH1      |                   |                             |

**Example:**

```
/* tmr2 flexible function enable */
dma_flexible_config(DMA1, FLEX_CHANNEL1, DMA_FLEXIBLE_TMR2_OVERFLOW);
```

### 5.7.9 dma\_flag\_get function

The table below describes the function `dma_flag_get`.

Table 151. `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**

The 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         |

**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);
}
```

## 5.7.10 dma\_flag\_clear function

The table below describes the function `dma_flag_clear`.

**Table 152. `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      | <code>dmax_flag</code> : 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 and check its status, including:

|                                |                                            |
|--------------------------------|--------------------------------------------|
| <code>DMA1_GL1_FLAG:</code>    | DMA1 channel 1 global flag                 |
| <code>DMA1_FDT1_FLAG:</code>   | DMA1 channel 1 transfer complete flag      |
| <code>DMA1_HDT1_FLAG:</code>   | DMA1 channel 1 half transfer complete flag |
| <code>DMA1_DTERR1_FLAG:</code> | DMA1 channel 1 transfer error flag         |
| <code>DMA1_GL2_FLAG:</code>    | DMA1 channel 2 global flag                 |
| <code>DMA1_FDT2_FLAG:</code>   | DMA1 channel 2 transfer complete flag      |
| <code>DMA1_HDT2_FLAG:</code>   | DMA1 channel 2 half transfer complete flag |
| <code>DMA1_DTERR2_FLAG:</code> | DMA1 channel 2 transfer error flag         |
| <code>DMA1_GL3_FLAG:</code>    | DMA1 channel 3 global flag                 |
| <code>DMA1_FDT3_FLAG:</code>   | DMA1 channel 3 transfer complete flag      |
| <code>DMA1_HDT3_FLAG:</code>   | DMA1 channel 3 half transfer complete flag |
| <code>DMA1_DTERR3_FLAG:</code> | DMA1 channel 3 transfer error flag         |
| <code>DMA1_GL4_FLAG:</code>    | DMA1 channel 4 global flag                 |
| <code>DMA1_FDT4_FLAG:</code>   | DMA1 channel 4 transfer complete flag      |
| <code>DMA1_HDT4_FLAG:</code>   | DMA1 channel 4 half transfer complete flag |
| <code>DMA1_DTERR4_FLAG:</code> | DMA1 channel 4 transfer error flag         |
| <code>DMA1_GL5_FLAG:</code>    | DMA1 channel 5 global flag                 |
| <code>DMA1_FDT5_FLAG:</code>   | DMA1 channel 5 transfer complete flag      |
| <code>DMA1_HDT5_FLAG:</code>   | DMA1 channel 5 half transfer complete flag |
| <code>DMA1_DTERR5_FLAG:</code> | DMA1 channel 5 transfer error flag         |
| <code>DMA1_GL6_FLAG:</code>    | DMA1 channel 6 global flag                 |
| <code>DMA1_FDT6_FLAG:</code>   | DMA1 channel 6 transfer complete flag      |
| <code>DMA1_HDT6_FLAG:</code>   | DMA1 channel 6 half transfer complete flag |
| <code>DMA1_DTERR6_FLAG:</code> | DMA1 channel 6 transfer error flag         |
| <code>DMA1_GL7_FLAG:</code>    | DMA1 channel 7 global flag                 |
| <code>DMA1_FDT7_FLAG:</code>   | DMA1 channel 7 transfer complete flag      |
| <code>DMA1_HDT7_FLAG:</code>   | DMA1 channel 7 half transfer complete flag |
| <code>DMA1_DTERR7_FLAG:</code> | DMA1 channel 7 transfer error flag         |

**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);
 dma_flag_clear(DMA2_FDT1_FLAG);
}
```

## 5.8 Real-time clock (ERTC)

The ERTC register structure ertc\_type is defined in the “at32f425\_ertc.h”:

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

} ertc_type;
```

The table below gives a list of the ERTC registers:

**Table 153. Summary of ERTC registers**

| Register | Description                               |
|----------|-------------------------------------------|
| time     | ERTC time register                        |
| date     | ERTC date register                        |
| ctrl     | ERTC control register                     |
| sts      | ERTC initialization and status register   |
| div      | ERTC divider register                     |
| wat      | ERTC wakeup timer register                |
| ala      | ERTC alarm clock A register               |
| wp       | ERTC write protection register            |
| sbs      | ERTC subsecond register                   |
| tadj     | ERTC time adjustment register             |
| tstm     | ERTC time stamp time register             |
| tsdt     | ERTC time stamp date register             |
| tssbs    | ERTC time stamp subsecond register        |
| scal     | ERTC smooth calibration register          |
| tamp     | ERTC tamper configuration register        |
| alasbs   | ERTC alarm clock A subsecond register     |
| bprx     | ERTC battery powered domain data register |

The table below gives a list of ERTC library functions.

**Table 154. Summary of ERTC library functions**

| Function name              | Description                       |
|----------------------------|-----------------------------------|
| ertc_num_to_bcd            | Convert number to BCD code        |
| ertc_bcd_to_num            | Convert BCD code to number        |
| ertc_write_protect_enable  | Enable write protection           |
| ertc_write_protect_disable | Disable write protection          |
| ertc_wait_update           | Wait for register update complete |
| ertc_wait_flag             | Wait flag                         |
| ertc_init_mode_enter       | Enter initialization mode         |
| ertc_init_mode_exit        | Exit initialization mode          |

|                                |                                                                           |
|--------------------------------|---------------------------------------------------------------------------|
| ertc_reset                     | Reset ERTC registers                                                      |
| ertc_divider_set               | Divider setting                                                           |
| ertc_hour_mode_set             | Hour mode setting                                                         |
| ertc_date_set                  | Date setting                                                              |
| ertc_time_set                  | Time setting                                                              |
| ertc_calendar_get              | Get calendar                                                              |
| ertc_sub_second_get            | Get the current subsecond                                                 |
| ertc_alarm_mask_set            | Set alarm mask                                                            |
| ertc_alarm_week_date_select    | Alarm time format selection (week/date)                                   |
| ertc_alarm_set                 | Set alarm                                                                 |
| ertc_alarm_sub_second_set      | Set alarm subsecond                                                       |
| ertc_alarm_enable              | Enable alarm                                                              |
| ertc_alarm_get                 | Get alarm value                                                           |
| ertc_alarm_sub_second_get      | Get alarm subsecond                                                       |
| ertc_wakeup_clock_set          | Select wakeup clock source                                                |
| ertc_wakeup_counter_set        | Set wakeup counter value                                                  |
| ertc_wakeup_counter_get        | Get wakeup counter value                                                  |
| ertc_wakeup_enable             | Enable wakeup timer                                                       |
| ertc_smooth_calibration_config | Configure smooth calibration                                              |
| ertc_cal_output_select         | Calibration output source selection                                       |
| ertc_cal_output_enable         | Enable calibration output                                                 |
| ertc_time_adjust               | Time adjustment                                                           |
| ertc_daylight_set              | Set daylight saving time                                                  |
| ertc_daylight_bpr_get          | Get daylight saving time battery powered domain data register value (BPR) |
| ertc_refer_clock_detect_enable | Enable reference clock detection                                          |
| ertc_direct_read_enable        | Enable direct read mode                                                   |
| ertc_output_set                | Set event output                                                          |
| ertc_timestamp_valid_edge_set  | Set time stamp detection valid edge                                       |
| ertc_timestamp_enable          | Enable time stamp                                                         |
| ertc_timestamp_get             | Get time stamp                                                            |
| ertc_timestamp_sub_second_get  | Get time stamp subsecond                                                  |
| ertc_tamper_pull_up_enable     | Enable tamper pin pull-up resistor                                        |
| ertc_tamper_precharge_set      | Set tamper pin precharge time                                             |
| ertc_tamper_filter_set         | Set tamper filter time                                                    |
| ertc_tamper_detect_freq_set    | Set tamper detection frequency                                            |
| ertc_tamper_valid_edge_set     | Set tamper detection valid edge                                           |
| ertc_tamper_timestamp_enable   | Enable time stamp upon a tamper event                                     |
| ertc_tamper_enable             | Enable tamper detection                                                   |
| ertc_interrupt_enable          | Enable interrupts                                                         |
| ertc_interrupt_get             | Get the status of interrupt enable                                        |
| ertc_flag_get                  | Get flag status                                                           |
| ertc_flag_clear                | Clear flag                                                                |
| ertc_bpr_data_write            | Write data to battery powered data register (BPR)                         |
| ertc_bpr_data_read             | Read from battery powered data register (BPR)                             |

## 5.8.1 ertc\_num\_to\_bcd function

The table below describes the function ertc\_num\_to\_bcd.

Table 155. ertc\_num\_to\_bcd function

| Name                   | Description                           |
|------------------------|---------------------------------------|
| Function name          | ertc_num_to_bcd                       |
| Function prototype     | uint8_t ertc_num_to_bcd(uint8_t num); |
| Function description   | Convert number into BCD format        |
| Input parameter 1      | num: number to be converted           |
| Output parameter       | NA                                    |
| Return value           | BCD code                              |
| Required preconditions | NA                                    |
| Called functions       | NA                                    |

**Example:**

```
ertc_num_to_bcd(12);
```

## 5.8.2 ertc\_bcd\_to\_num function

The table below describes the function ertc\_bcd\_to\_num.

Table 156. ertc\_bcd\_to\_num function

| Name                   | Description                                 |
|------------------------|---------------------------------------------|
| Function name          | ertc_bcd_to_num                             |
| Function prototype     | uint8_t ertc_bcd_to_num(uint8_t bcd);       |
| Function description   | Convert BCD code into number                |
| Input parameter 1      | bcd: BCD code to be converted               |
| Output parameter       | NA                                          |
| Return value           | Return the number corresponding to BCD code |
| Required preconditions | NA                                          |
| Called functions       | NA                                          |

**Example:**

```
ertc_bcd_to_num(0x12);
```

### 5.8.3 ertc\_write\_protect\_enable function

The table below describes the function ertc\_write\_protect\_enable.

**Table 157. ertc\_write\_protect\_enable function**

| Name                   | Description                           |
|------------------------|---------------------------------------|
| Function name          | ertc_write_protect_enable             |
| Function prototype     | void ertc_write_protect_enable(void); |
| Function description   | Write protection enable               |
| Input parameter 1      | NA                                    |
| Output parameter       | NA                                    |
| Return value           | NA                                    |
| Required preconditions | NA                                    |
| Called functions       | NA                                    |

**Example:**

```
ertc_write_protect_enable();
```

### 5.8.4 ertc\_write\_protect\_disable function

The table below describes the function ertc\_write\_protect\_disable.

**Table 158. ertc\_write\_protect\_disable function**

| Name                   | Description                            |
|------------------------|----------------------------------------|
| Function name          | ertc_write_protect_disable             |
| Function prototype     | void ertc_write_protect_disable(void); |
| Function description   | Write protection disable               |
| Input parameter 1      | NA                                     |
| Output parameter       | NA                                     |
| Return value           | NA                                     |
| Required preconditions | NA                                     |
| Called functions       | NA                                     |

**Example:**

```
ertc_write_protect_disable();
```

## 5.8.5 ertc\_wait\_update function

The table below describes the function ertc\_wait\_update.

**Table 159. ertc\_wait\_update function**

| Name                   | Description                                                   |
|------------------------|---------------------------------------------------------------|
| Function name          | ertc_wait_update                                              |
| Function prototype     | error_status ertc_wait_update(void);                          |
| Function description   | Wait for register to finish update                            |
| Input parameter 1      | NA                                                            |
| Output parameter       | NA                                                            |
| Return value           | SUCCESS: register update complete<br>ERROR: flag wait timeout |
| Required preconditions | NA                                                            |
| Called functions       | NA                                                            |

**Example:**

|                     |
|---------------------|
| ertc_wait_update(); |
|---------------------|

## 5.8.6 ertc\_wait\_flag function

The table below describes the function ertc\_wait\_flag.

**Table 160. ertc\_wait\_flag function**

| Name                   | Description                                                                                                                                             |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | ertc_wait_flag                                                                                                                                          |
| Function prototype     | error_status ertc_wait_flag(uint32_t flag, flag_status status);                                                                                         |
| Function description   | Wait flag                                                                                                                                               |
| Input parameter 1      | flag: flag selection<br>Refer to the "flag" description below for details.                                                                              |
| Input parameter 1      | status: flag status.<br>After the flag status is set, the function remains stuck here until flag status changes.<br>This parameter can be SET or RESET. |
| Output parameter       | NA                                                                                                                                                      |
| Return value           | SUCCESS: flag state changed<br>ERROR: flag wait timeout                                                                                                 |
| Required preconditions | NA                                                                                                                                                      |
| Called functions       | NA                                                                                                                                                      |

### flag

Flag selection

- ERTC\_ALAWF\_FLAG: Alarm A write enable flag
- ERTC\_WATWF\_FLAG: Wakeup timer register write enable flag
- ERTC\_TADJF\_FLAG: Time adjustment flag
- ERTC\_CALUPDF\_FLAG: Calibration value update complete flag

**Example:**

|                                         |
|-----------------------------------------|
| ertc_wait_flag(ERTC_ALAWF_FLAG, RESET); |
|-----------------------------------------|

## 5.8.7 ertc\_init\_mode\_enter function

The table below describes the function ertc\_init\_mode\_enter.

**Table 161. ertc\_init\_mode\_enter function**

| Name                   | Description                                                            |
|------------------------|------------------------------------------------------------------------|
| Function name          | ertc_init_mode_enter                                                   |
| Function prototype     | error_status ertc_init_mode_enter(void);                               |
| Function description   | Enter initialization mode                                              |
| Input parameter 1      | NA                                                                     |
| Output parameter       | NA                                                                     |
| Return value           | SUCCESS: Initialization mode is entered successfully<br>ERROR: Timeout |
| Required preconditions | NA                                                                     |
| Called functions       | NA                                                                     |

**Example:**

```
ertc_init_mode_enter();
```

## 5.8.8 ertc\_init\_mode\_exit function

The table below describes the function ertc\_init\_mode\_exit.

**Table 162. ertc\_init\_mode\_exit function**

| Name                   | Description                     |
|------------------------|---------------------------------|
| Function name          | ertc_init_mode_exit             |
| Function prototype     | void ertc_init_mode_exit(void); |
| Function description   | Exit initialization mode        |
| Input parameter 1      | NA                              |
| Output parameter       | NA                              |
| Return value           | NA                              |
| Required preconditions | NA                              |
| Called functions       | NA                              |

**Example:**

```
ertc_init_mode_exit();
```

## 5.8.9 ertc\_reset function

The table below describes the function ertc\_reset.

**Table 163. ertc\_reset function**

| Name                   | Description                                      |
|------------------------|--------------------------------------------------|
| Function name          | ertc_reset                                       |
| Function prototype     | error_status ertc_reset(void);                   |
| Function description   | Reset all ERTC registers                         |
| Input parameter 1      | NA                                               |
| Output parameter       | NA                                               |
| Return value           | SUCCESS: Reset successful<br>ERROR: Reset failed |
| Required preconditions | NA                                               |
| Called functions       | NA                                               |

**Example:**

```
ertc_reset();
```

## 5.8.10 ertc\_divider\_set function

The table below describes the function ertc\_divider\_set.

**Table 164. ertc\_divider\_set function**

| Name                   | Description                                                                                                                                                                                  |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | ertc_divider_set                                                                                                                                                                             |
| Function prototype     | error_status ertc_divider_set(uint16_t div_a, uint16_t div_b);                                                                                                                               |
| Function description   | Divider settings, frequency division value $(div\_a + 1) * (div\_b + 1) = ERTC\_CLK$ frequency<br>For example, if 32768Hz is used, the frequency division should be div_a = 127, div_b = 255 |
| Input parameter 1      | div_a: divider A, range: 0~0x7F                                                                                                                                                              |
| Input parameter 2      | div_b: divider B, range: 0~0x7FFF                                                                                                                                                            |
| Output parameter       | NA                                                                                                                                                                                           |
| Return value           | SUCCESS: Reset successful<br>ERROR: Reset failed                                                                                                                                             |
| Required preconditions | NA                                                                                                                                                                                           |
| Called functions       | NA                                                                                                                                                                                           |

**Example:**

```
ertc_divider_set(127, 255);
```

## 5.8.11 ertc\_hour\_mode\_set function

The table below describes the function ertc\_hour\_mode\_set.

**Table 165. ertc\_hour\_mode\_set function**

| Name                   | Description                                                               |
|------------------------|---------------------------------------------------------------------------|
| Function name          | ertc_hour_mode_set                                                        |
| Function prototype     | error_status ertc_hour_mode_set(ertc_hour_mode_set_type mode);            |
| Function description   | Hour mode settings                                                        |
| Input parameter 1      | mode: hour mode<br>Refer to the following description “Mode” for details. |
| Output parameter       | NA                                                                        |
| Return value           | SUCCESS: Setting success<br>ERROR: Setting error                          |
| Required preconditions | NA                                                                        |
| Called functions       | NA                                                                        |

**mode**

ERTC\_HOUR\_MODE\_24: 24-hour format

ERTC\_HOUR\_MODE\_12: 12-hour format

**Example:**

|                                        |
|----------------------------------------|
| ertc_hour_mode_set(ERTC_HOUR_MODE_24); |
|----------------------------------------|

## 5.8.12 ertc\_date\_set function

The table below describes the function ertc\_date\_set.

**Table 166. ertc\_date\_set function**

| Name                   | Description                                                                          |
|------------------------|--------------------------------------------------------------------------------------|
| Function name          | ertc_date_set                                                                        |
| Function prototype     | error_status ertc_date_set(uint8_t year, uint8_t month, uint8_t date, uint8_t week); |
| Function description   | Set date: year, month, date, weekday                                                 |
| Input parameter 1      | year: range 0~99                                                                     |
| Input parameter 2      | month: range 1~12                                                                    |
| Input parameter3       | date: range 1~31                                                                     |
| Input parameter4       | week: range 1~7                                                                      |
| Output parameter       | NA                                                                                   |
| Return value           | SUCCESS: Setting success<br>ERROR: Setting error                                     |
| Required preconditions | NA                                                                                   |
| Called functions       | NA                                                                                   |

**Example:**

|                              |
|------------------------------|
| ertc_date_set(22, 5, 26, 4); |
|------------------------------|

## 5.8.13 ertc\_time\_set function

The table below describes the function ertc\_time\_set.

**Table 167. ertc\_time\_set function**

| Name                   | Description                                                                                                                                     |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | ertc_time_set                                                                                                                                   |
| Function prototype     | error_status ertc_time_set(uint8_t hour, uint8_t min, uint8_t sec, ertc_am_pm_type ampm);                                                       |
| Function description   | Set time: hour, minute, second, AM/PM (for 12-hour format only)                                                                                 |
| Input parameter 1      | hour: range 0~23                                                                                                                                |
| Input parameter 2      | min: range 0~59                                                                                                                                 |
| Input parameter3       | sec: range 0~59                                                                                                                                 |
| Input parameter4       | ampm: AM/PM in 12-hour format (for 12-hour format only, don't care in 24-hour format)<br>Refer to the following description "ampm" for details. |
| Output parameter       | NA                                                                                                                                              |
| Return value           | SUCCESS: Setting success<br>ERROR: Setting error                                                                                                |
| Required preconditions | NA                                                                                                                                              |
| Called functions       | NA                                                                                                                                              |

### ampm

AM/PM in 12-hour format (for 12-hour format only, don't care in 24-hour format)

ERTC\_24H: 24-hour format (for 24-hour format)

ERTC\_AM: AM in 12-hour format

ERTC\_PM: PM in 12-hour format

### Example:

```
ertc_time_set(12, 1, 20, ERTC_24H);
```

## 5.8.14 ertc\_calendar\_get function

The table below describes the function ertc\_calendar\_get.

**Table 168. ertc\_calendar\_get function**

| Name                   | Description                                                                     |
|------------------------|---------------------------------------------------------------------------------|
| Function name          | ertc_calendar_get                                                               |
| Function prototype     | void ertc_calendar_get(ertc_time_type* time);                                   |
| Function description   | Get calendar, including year, month, date, weekday, hour, minute, second, AM/PM |
| Input parameter 1      | time: ertc_time_type pointer                                                    |
| Output parameter       | NA                                                                              |
| Return value           | NA                                                                              |
| Required preconditions | NA                                                                              |
| Called functions       | NA                                                                              |

ertc\_time\_type\* time

The ertc\_time\_type is defined in the "at32f425\_ertc.h":

typedef struct

{

```

 uint8_t year;
 uint8_t month;
 uint8_t day;
 uint8_t hour;
 uint8_t min;
 uint8_t sec;
 uint8_t week;
 ertc_am_pm_type ampm;
} ertc_time_type;
year
Range 0~99
month
Range 1~12
day
Range 1~31
week
Range 1~7
hour
Range 0~23
min
Range 0~59
sec
Range 0~59
ampm
AM/PM in 12-hour format (for 12-hour format only, doesn't care in 24 hour), including:

ERTC_AM: AM in 12 hour format

ERTC_PM: PM in 12 hour format

```

**Example:**

`ertc_calendar_get(&time);`

## 5.8.15 ertc\_sub\_second\_get function

The table below describes the function ertc\_sub\_second\_get.

**Table 169. ertc\_sub\_second\_get function**

| Name                   | Description                                            |
|------------------------|--------------------------------------------------------|
| Function name          | ertc_sub_second_get                                    |
| Function prototype     | uint32_t ertc_sub_second_get(void);                    |
| Function description   | Get current subsecond (the current value of divider B) |
| Input parameter 1      | NA                                                     |
| Output parameter       | NA                                                     |
| Return value           | Current subsecond                                      |
| Required preconditions | NA                                                     |
| Called functions       | NA                                                     |

**Example:**

`ertc_sub_second_get();`

## 5.8.16 ertc\_alarm\_mask\_set function

The table below describes the function ertc\_alarm\_mask\_set.

Table 170. ertc\_alarm\_mask\_set function

| Name                   | Description                                                                           |
|------------------------|---------------------------------------------------------------------------------------|
| Function name          | ertc_alarm_mask_set                                                                   |
| Function prototype     | void ertc_alarm_mask_set(ertc_alarm_type alarm_x, uint32_t mask);                     |
| Function description   | Set alarm mask                                                                        |
|                        | alarm_x: alarm selection<br>Refer to the following description "alarm_x" for details. |
| Input parameter 1      | mask: Set alarm mask<br>Refer to the following description "mask" for details.        |
| Output parameter       | NA                                                                                    |
| Return value           | NA                                                                                    |
| Required preconditions | NA                                                                                    |
| Called functions       | NA                                                                                    |

### alarm\_x

Alarm selection

ERTC\_ALA: Alarm A

### mask

Set alarm mask

|                            |                                              |
|----------------------------|----------------------------------------------|
| ERTC_ALARM_MASK_NONE:      | No mask, alarm is relevant to all fields     |
| ERTC_ALARM_MASK_SEC:       | Mask second, alarm is not relevant to second |
| ERTC_ALARM_MASK_MIN:       | Mask minute, alarm is not relevant to minute |
| ERTC_ALARM_MASK_HOUR:      | Mask hour, alarm is not relevant to hour     |
| ERTC_ALARM_MASK_DATE_WEEK: | Mask date, alarm is not relevant to date     |
| ERTC_ALARM_MASK_ALL:       | Mask all. Generate an alarm per one second   |

### Example:

```
ertc_alarm_mask_set(ERTC_ALA, ERTC_ALARM_MASK_NONE);
```

## 5.8.17 ertc\_alarm\_week\_date\_select function

The table below describes the function ertc\_alarm\_week\_date\_select.

Table 171. ertc\_alarm\_week\_date\_select function

| Name                   | Description                                                                                  |
|------------------------|----------------------------------------------------------------------------------------------|
| Function name          | ertc_alarm_week_date_select                                                                  |
| Function prototype     | void ertc_alarm_week_date_select(ertc_alarm_type alarm_x,<br>ertc_week_date_select_type wk); |
| Function description   | Alarm time format selection: week/date                                                       |
| Input parameter 1      | alarm_x: alarm selection<br>Refer to the following description “alarm_x” for details.        |
| Input parameter 2      | wk: alarm week/date format selection<br>Refer to the following description “wk” for details. |
| Output parameter       | NA                                                                                           |
| Return value           | NA                                                                                           |
| Required preconditions | NA                                                                                           |
| Called functions       | NA                                                                                           |

### alarm\_x

Alarm selection

ERTC\_ALA: Alarm A

### wk

Alarm week/date format selection

ERTC\_SELECT\_DATE: Date mode

ERTC\_SELECT\_WEEK: Week mode

### Example:

```
ertc_alarm_week_date_select(ERTC_ALA, ERTC_SELECT_DATE);
```

## 5.8.18 ertc\_alarm\_set function

The table below describes the function ertc\_alarm\_set.

**Table 172. ertc\_alarm\_set function**

| Name                   | Description                                                                                                                                   |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | ertc_alarm_set                                                                                                                                |
| Function prototype     | void ertc_alarm_set(ertc_alarm_type alarm_x, uint8_t week_date, uint8_t hour, uint8_t min, uint8_t sec, ertc_am_pm_type ampm);                |
| Function description   | Set alarm                                                                                                                                     |
| Input parameter 1      | alarm_x: alarm selection<br>Refer to the following description “alarm_x” for details.                                                         |
| Input parameter 2      | week_date: date or week, depending on the ertc_alarm_week_date_select()<br>Date: range 1~31<br>Week: range 1~7                                |
| Input parameter3       | hour: range 0~23                                                                                                                              |
| Input parameter4       | min: range 0~59                                                                                                                               |
| Input parameter5       | sec: range 0~59                                                                                                                               |
| Input parameter6       | ampm: AM/PM in 12-hour format (12 hour format only, doesn't care in 24-hour format)<br>Refer to the following description “ampm” for details. |
| Output parameter       | NA                                                                                                                                            |
| Return value           | NA                                                                                                                                            |
| Required preconditions | NA                                                                                                                                            |
| Called functions       | NA                                                                                                                                            |

### **alarm\_x**

Alarm selection

ERTC\_ALA: Alarm A

### **ampm**

AM/PM in 12-hour format (for 12 hour format only, doesn't care in 24 hour)

ERTC\_24H: 24-hour format (for 24 hour format)

ERTC\_AM: AM in 12-hour format

ERTC\_PM: PM in 12-hour format

### **Example:**

```
ertc_alarm_set(ERTC_ALA, 15, 8, 0, 0, ERTC_24H);
```

## 5.8.19 ertc\_alarm\_sub\_second\_set function

The table below describes the function ertc\_alarm\_sub\_second\_set.

**Table 173. ertc\_alarm\_sub\_second\_set function**

| Name                   | Description                                                                                             |
|------------------------|---------------------------------------------------------------------------------------------------------|
| Function name          | ertc_alarm_sub_second_set                                                                               |
| Function prototype     | void ertc_alarm_sub_second_set(ertc_alarm_type alarm_x, uint32_t value, ertc_alarm_sbs_mask_type mask); |
| Function description   | Set alarm subsecond                                                                                     |
| Input parameter 1      | alarm_x: alarm selection<br>Refer to the following description “alarm_x” for details.                   |
| Input parameter 2      | value: subsecond value, range 0~0x7FFF                                                                  |
| Input parameter3       | mask: alarm mask settings<br>Refer to the following description “mask” for details.                     |
| Output parameter       | NA                                                                                                      |
| Return value           | NA                                                                                                      |
| Required preconditions | NA                                                                                                      |
| Called functions       | NA                                                                                                      |

### alarm\_x

Alarm selection

ERTC\_ALA: Alarm A

### mask

Subsecond mask

|                            |                           |
|----------------------------|---------------------------|
| ERTC_ALARM_SBS_MASK_ALL:   | Mask all                  |
| ERTC_ALARM_SBS_MASK_14_1:  | Only match SBS bit [0]    |
| ERTC_ALARM_SBS_MASK_14_2:  | Only match SBS bit [1:0]  |
| ERTC_ALARM_SBS_MASK_14_3:  | Only match SBS bit [2:0]  |
| ERTC_ALARM_SBS_MASK_14_4:  | Only match SBS bit [3:0]  |
| ERTC_ALARM_SBS_MASK_14_5:  | Only match SBS bit [4:0]  |
| ERTC_ALARM_SBS_MASK_14_6:  | Only match SBS bit [5:0]  |
| ERTC_ALARM_SBS_MASK_14_7:  | Only match SBS bit [6:0]  |
| ERTC_ALARM_SBS_MASK_14_8:  | Only match SBS bit [7:0]  |
| ERTC_ALARM_SBS_MASK_14_9:  | Only match SBS bit [8:0]  |
| ERTC_ALARM_SBS_MASK_14_10: | Only match SBS bit [9:0]  |
| ERTC_ALARM_SBS_MASK_14_11: | Only match SBS bit [10:0] |
| ERTC_ALARM_SBS_MASK_14_12: | Only match SBS bit [11:0] |
| ERTC_ALARM_SBS_MASK_14_13: | Only match SBS bit [12:0] |
| ERTC_ALARM_SBS_MASK_14:    | Only match SBS bit [13:0] |
| ERTC_ALARM_SBS_MASK_NONE:  | Only match SBS bit [14:0] |

### Example:

```
ertc_alarm_sub_second_set(ERTC_ALA, 200, ERTC_ALARM_SBS_MASK_NONE);
```

## 5.8.20 ertc\_alarm\_enable function

The table below describes the function ertc\_alarm\_enable.

**Table 174. ertc\_alarm\_enable function**

| Name                   | Description                                                                           |
|------------------------|---------------------------------------------------------------------------------------|
| Function name          | ertc_alarm_enable                                                                     |
| Function prototype     | error_status ertc_alarm_enable(ertc_alarm_type alarm_x, confirm_state new_state);     |
| Function description   | Alarm enable                                                                          |
| Input parameter 1      | alarm_x: alarm selection<br>Refer to the following description “alarm_x” for details. |
| Input parameter 2      | new_state: alarm enable status<br>This parameter can be TRUE or FALSE.                |
| Output parameter       | NA                                                                                    |
| Return value           | SUCCESS: Setting success<br>ERROR: Setting error                                      |
| Required preconditions | NA                                                                                    |
| Called functions       | NA                                                                                    |

### alarm\_x

Alarm selection

ERTC\_ALA: Alarm A

#### Example:

```
ertc_alarm_enable(ERTC_ALA, TRUE);
```

## 5.8.21 ertc\_alarm\_get function

The table below describes the function ertc\_alarm\_get.

**Table 175. ertc\_alarm\_get function**

| Name                   | Description                                                                           |
|------------------------|---------------------------------------------------------------------------------------|
| Function name          | ertc_alarm_get                                                                        |
| Function prototype     | void ertc_alarm_get(ertc_alarm_type alarm_x, ertc_alarm_value_type* alarm);           |
| Function description   | Get alarm value                                                                       |
| Input parameter 1      | alarm_x: alarm selection<br>Refer to the following description “alarm_x” for details. |
| Input parameter 2      | alarm: ertc_alarm_value_type pointer                                                  |
| Output parameter       | NA                                                                                    |
| Return value           | NA                                                                                    |
| Required preconditions | NA                                                                                    |
| Called functions       | NA                                                                                    |

### alarm\_x

Alarm selection

ERTC\_ALA: Alarm A

ertc\_alarm\_value\_type\* alarm

ertc\_alarm\_value\_type is defined in the “at32f425\_ertc.h”:

typedef struct

```
{
 uint8_t day;
 uint8_t hour;
 uint8_t min;
 uint8_t sec;
 ertc_am_pm_type ampm;
 uint32_t mask;
 uint8_t week_date_sel;
 uint8_t week;
} ertc_alarm_value_type;
```

**day**

Range 1~31

**hour**

Range 0~23

**min**

Range 0~59

**sec**

Range 0~59

**ampm**

AM/PM in 12-hour format (for 12-hour format only, doesn't care in 24 hour), including:

ERTC\_AM: AM in 12 hour format

ERTC\_PM: PM in 12 hour format

**mask**

Alarm mask value, including:

|                            |             |
|----------------------------|-------------|
| ERTC_ALARM_MASK_NONE:      | No mask     |
| ERTC_ALARM_MASK_SEC:       | Mask second |
| ERTC_ALARM_MASK_MIN:       | Mask minute |
| ERTC_ALARM_MASK_HOUR:      | Mask hour   |
| ERTC_ALARM_MASK_DATE_WEEK: | Mask date   |
| ERTC_ALARM_MASK_ALL:       | Mask all    |

**week\_date\_sel**

Alarm week/date format, including:

ERTC\_SELECT\_DATE: date mode

ERTC\_SELECT\_WEEK: week mode

**week**

Range 1~7

**Example:**

```
ertc_alarm_get(ERTC_ALA, &alarm);
```

## 5.8.22 ertc\_alarm\_sub\_second\_get function

The table below describes the function ertc\_alarm\_sub\_second\_get.

**Table 176. ertc\_alarm\_sub\_second\_get function**

| Name                   | Description                                                                           |
|------------------------|---------------------------------------------------------------------------------------|
| Function name          | ertc_alarm_sub_second_get                                                             |
| Function prototype     | uint32_t ertc_alarm_sub_second_get(ertc_alarm_type alarm_x);                          |
| Function description   | Get alarm subsecond value                                                             |
| Input parameter 1      | alarm_x: alarm selection<br>Refer to the following description "alarm_x" for details. |
| Output parameter       | NA                                                                                    |
| Return value           | Alarm subsecond value                                                                 |
| Required preconditions | NA                                                                                    |
| Called functions       | NA                                                                                    |

### alarm\_x

Alarm selection

ERTC\_ALA: Alarm A

#### Example:

```
ertc_alarm_sub_second_get(ERTC_ALA);
```

## 5.8.23 ertc\_wakeup\_clock\_set function

The table below describes the function ertc\_wakeup\_clock\_set.

**Table 177. ertc\_wakeup\_clock\_set function**

| Name                   | Description                                                                                     |
|------------------------|-------------------------------------------------------------------------------------------------|
| Function name          | ertc_wakeup_clock_set                                                                           |
| Function prototype     | void ertc_wakeup_clock_set(ertc_wakeup_clock_type clock);                                       |
| Function description   | Select wakeup timer clock source                                                                |
| Input parameter 1      | clock: clock source for wakeup timer<br>Refer to the following description "clock" for details. |
| Output parameter       | NA                                                                                              |
| Return value           | NA                                                                                              |
| Required preconditions | NA                                                                                              |
| Called functions       | NA                                                                                              |

### clock

Wakeup timer clock source

ERTC\_WAT\_CLK\_ERTCCLK\_DIV16: ERTC\_CLK / 16

ERTC\_WAT\_CLK\_ERTCCLK\_DIV8: ERTC\_CLK / 8

ERTC\_WAT\_CLK\_ERTCCLK\_DIV4: ERTC\_CLK / 4

ERTC\_WAT\_CLK\_ERTCCLK\_DIV2: ERTC\_CLK / 2

ERTC\_WAT\_CLK\_CK\_B\_16BITS: CK\_B (1Hz calendar clock), wakeup counter value = ERTC\_WAT

ERTC\_WAT\_CLK\_CK\_B\_17BITS: CK\_B (1Hz calendar clock), wakeup counter value = ERTC\_WAT + 65535

#### Example:

```
ertc_wakeup_clock_set(ERTC_WAT_CLK_CK_B_16BITS);
```

## 5.8.24 ertc\_wakeup\_counter\_set function

The table below describes the function ertc\_wakeup\_counter\_set.

**Table 178. ertc\_wakeup\_counter\_set function**

| Name                   | Description                                     |
|------------------------|-------------------------------------------------|
| Function name          | ertc_wakeup_counter_set                         |
| Function prototype     | void ertc_wakeup_counter_set(uint32_t counter); |
| Function description   | Set wakeup counter value                        |
| Input parameter 1      | counter: ounter value, range 0~0xFFFF           |
| Output parameter       | NA                                              |
| Return value           | NA                                              |
| Required preconditions | NA                                              |
| Called functions       | NA                                              |

**Example:**

```
ertc_wakeup_counter_set(0x7FFF);
```

## 5.8.25 ertc\_wakeup\_counter\_get function

The table below describes the function ertc\_wakeup\_counter\_get.

**Table 179. ertc\_wakeup\_counter\_get function**

| Name                   | Description                             |
|------------------------|-----------------------------------------|
| Function name          | ertc_wakeup_counter_get                 |
| Function prototype     | uint16_t ertc_wakeup_counter_get(void); |
| Function description   | Get the current wakeup counter value    |
| Input parameter 1      | NA                                      |
| Output parameter       | NA                                      |
| Return value           | Return the current wakeup counter value |
| Required preconditions | NA                                      |
| Called functions       | NA                                      |

**Example:**

```
ertc_wakeup_counter_get();
```

## 5.8.26 ertc\_wakeup\_enable function

The table below describes the function ertc\_wakeup\_enable.

**Table 180. ertc\_wakeup\_enable function**

| Name                   | Description                                                                  |
|------------------------|------------------------------------------------------------------------------|
| Function name          | ertc_wakeup_enable                                                           |
| Function prototype     | error_status ertc_wakeup_enable(confirm_state new_state);                    |
| Function description   | Enable wakeup timer                                                          |
| Input parameter 1      | new_state: wakeup timer enable state<br>This parameter can be TRUE or FALSE. |
| Output parameter       | NA                                                                           |
| Return value           | SUCCESS: Set success<br>ERROR: Set error                                     |
| Required preconditions | NA                                                                           |
| Called functions       | NA                                                                           |

**Example:**

```
ertc_wakeup_enable(TRUE);
```

## 5.8.27 ertc\_smooth\_calibration\_config function

The table below describes the function ertc\_smooth\_calibration\_config.

**Table 181. ertc\_smooth\_calibration\_config function**

| Name                   | Description                                                                                                                              |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | ertc_smooth_calibration_config                                                                                                           |
| Function prototype     | error_status ertc_smooth_calibration_config(ertc_smooth_cal_period_type period, ertc_smooth_cal_clk_add_type clk_add, uint32_t clk_dec); |
| Function description   | et smooth digital calibration                                                                                                            |
| Input parameter 1      | period: calibration period<br>Refer to the following "period" descriptions for details.                                                  |
| Input parameter 2      | clk_add: add ERTC CLK cycles<br>Refer to the following "clk_add" descriptions for details.                                               |
| Input parameter3       | clk_dec: reduce ERTC CLK cycles, range 0~511                                                                                             |
| Output parameter       | NA                                                                                                                                       |
| Return value           | SUCCESS: Set success<br>ERROR: Set error                                                                                                 |
| Required preconditions | NA                                                                                                                                       |
| Called functions       | NA                                                                                                                                       |

### period

Calibration periods

ERTC\_SMOOTH\_CAL\_PERIOD\_32: 32 seconds

ERTC\_SMOOTH\_CAL\_PERIOD\_16: 16 seconds

ERTC\_SMOOTH\_CAL\_PERIOD\_8: 8 seconds

### clk\_add

Add ERTC CLK

ERTC\_SMOOTH\_CAL\_CLK\_ADD\_0: No effect

ERTC\_SMOOTH\_CAL\_CLK\_ADD\_512: Add 512 ERTC\_CLK cycles

**Example:**

```
ertc_smooth_calibration_config(ERTC_SMOOTH_CAL_PERIOD_32, ERTC_SMOOTH_CAL_CLK_ADD_0, 511);
```

### 5.8.28 ertc\_cal\_output\_select function

The table below describes the function ertc\_cal\_output\_select.

**Table 182. ertc\_cal\_output\_select function**

| Name                   | Description                                                                                    |
|------------------------|------------------------------------------------------------------------------------------------|
| Function name          | ertc_cal_output_select                                                                         |
| Function prototype     | void ertc_cal_output_select(ertc_cal_output_select_type output);                               |
| Function description   | Calibration output source selection                                                            |
| Input parameter 1      | output: Calibration output source<br>Refer to the following "output" descriptions for details. |
| Output parameter       | NA                                                                                             |
| Return value           | NA                                                                                             |
| Required preconditions | NA                                                                                             |
| Called functions       | NA                                                                                             |

**output**

Calibration output source

ERTC\_CAL\_OUTPUT\_512HZ: 512 Hz output

ERTC\_CAL\_OUTPUT\_1HZ: 1 Hz output

**Example:**

```
ertc_cal_output_select(ERTC_CAL_OUTPUT_1HZ);
```

### 5.8.29 ertc\_cal\_output\_enable function

The table below describes the function ertc\_cal\_output\_enable.

**Table 183. ertc\_cal\_output\_enable function**

| Name                   | Description                                                                        |
|------------------------|------------------------------------------------------------------------------------|
| Function name          | ertc_cal_output_enable                                                             |
| Function prototype     | void ertc_cal_output_enable(confirm_state new_state);                              |
| Function description   | Calibration output enable                                                          |
| Input parameter 1      | new_state: calibration output enable state<br>This parameter can be TRUE or FALSE. |
| Output parameter       | NA                                                                                 |
| Return value           | NA                                                                                 |
| Required preconditions | NA                                                                                 |
| Called functions       | NA                                                                                 |

**Example:**

```
ertc_cal_output_enable(TRUE);
```

## 5.8.30 ertc\_time\_adjust function

The table below describes the function ertc\_time\_adjust.

**Table 184. ertc\_time\_adjust function**

| Name                   | Description                                                                    |
|------------------------|--------------------------------------------------------------------------------|
| Function name          | ertc_time_adjust                                                               |
| Function prototype     | error_status ertc_time_adjust(ertc_time_adjust_type add1s, uint32_t decsbs);   |
| Function description   | Adjust time                                                                    |
| Input parameter 1      | add1s: add seconds<br>Refer to the following “add1s” descriptions for details. |
| Input parameter 2      | decsbs: reduce subseconds, range 0~0x7FFF                                      |
| Output parameter       | NA                                                                             |
| Return value           | SUCCESS: Set success<br>ERROR: Set error                                       |
| Required preconditions | NA                                                                             |
| Called functions       | NA                                                                             |

### add1s

This bit is used to add seconds.

ERTC\_TIME\_ADD\_NONE: No effect

ERTC\_TIME\_ADD\_1S: Add 1 second

### Example:

```
ertc_time_adjust(ERTC_TIME_ADD_1S, 254);
```

## 5.8.31 ertc\_daylight\_set function

The table below describes the function ertc\_daylight\_set.

**Table 185. ertc\_daylight\_set function**

| Name                   | Description                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------------------|
| Function name          | ertc_daylight_set                                                                                        |
| Function prototype     | void ertc_daylight_set(ertc_dst_operation_type operation, ertc_dst_save_type save);                      |
| Function description   | Set daylight-saving time                                                                                 |
| Input parameter 1      | operation: daylight-saving time settings<br>Refer to the following “operation” descriptions for details. |
| Input parameter 2      | save: save daylight time<br>Refer to the following “save” descriptions for details.                      |
| Output parameter       | NA                                                                                                       |
| Return value           | NA                                                                                                       |
| Required preconditions | NA                                                                                                       |
| Called functions       | NA                                                                                                       |

### operation

Daylight-saving time settings

ERTC\_DST\_ADD\_1H: Add 1 hour

ERTC\_DST\_DEC\_1H: Reduce 1 hour

### save

Save daylight time

ERTC\_DST\_SAVE\_0: set BPR bit to 0 in the CTRL register  
 ERTC\_DST\_SAVE\_1: set BPR bit to 1 in the CTRL register

**Example:**

```
ertc_daylight_set(ERTC_DST_ADD_1H, ERTC_DST_SAVE_1);
```

### 5.8.32 ertc\_daylight\_bpr\_get function

The table below describes the function ertc\_daylight\_bpr\_get.

**Table 186. ertc\_daylight\_bpr\_get function**

| Name                   | Description                                                                                      |
|------------------------|--------------------------------------------------------------------------------------------------|
| Function name          | ertc_daylight_bpr_get                                                                            |
| Function prototype     | uint8_t ertc_daylight_bpr_get(void);                                                             |
| Function description   | Get the value of daylight-saving time battery powered register (BPR bit in the CTRL register)    |
| Input parameter 1      | NA                                                                                               |
| Output parameter       | NA                                                                                               |
| Return value           | Return the value of daylight-saving time battery powered register (BPR bit in the CTRL register) |
| Required preconditions | NA                                                                                               |
| Called functions       | NA                                                                                               |

**Example:**

```
ertc_daylight_bpr_get();
```

### 5.8.33 ertc\_refer\_clock\_detect\_enable function

The table below describes the function ertc\_refer\_clock\_detect\_enable.

**Table 187. ertc\_refer\_clock\_detect\_enable function**

| Name                   | Description                                                                               |
|------------------------|-------------------------------------------------------------------------------------------|
| Function name          | ertc_refer_clock_detect_enable                                                            |
| Function prototype     | error_status ertc_refer_clock_detect_enable(confirm_state new_state);                     |
| Function description   | Enable reference clock detection                                                          |
| Input parameter 1      | new_state: reference clock detection enable state<br>This parameter can be TRUE or FALSE. |
| Output parameter       | NA                                                                                        |
| Return value           | SUCCESS: Set success<br>ERROR: Set error                                                  |
| Required preconditions | NA                                                                                        |
| Called functions       | NA                                                                                        |

**Example:**

```
ertc_refer_clock_detect_enable(TRUE);
```

### 5.8.34 ertc\_direct\_read\_enable function

The table below describes the function ertc\_direct\_read\_enable.

**Table 188. ertc\_direct\_read\_enable function**

| Name                   | Description                                                                      |
|------------------------|----------------------------------------------------------------------------------|
| Function name          | ertc_direct_read_enable                                                          |
| Function prototype     | void ertc_direct_read_enable(confirm_state new_state);                           |
| Function description   | Enable direct read mode                                                          |
| Input parameter 1      | new_state: direct read mode enable state<br>This parameter can be TRUE or FALSE. |
| Output parameter       | NA                                                                               |
| Return value           | NA                                                                               |
| Required preconditions | NA                                                                               |
| Called functions       | NA                                                                               |

**Example:**

```
ertc_direct_read_enable(TRUE);
```

### 5.8.35 ertc\_output\_set function

The table below describes the function ertc\_output\_set.

**Table 189. ertc\_output\_set function**

| Name                   | Description                                                                                                      |
|------------------------|------------------------------------------------------------------------------------------------------------------|
| Function name          | ertc_output_set                                                                                                  |
| Function prototype     | void ertc_output_set(ertc_output_source_type source, ertc_output_polarity_type polarity, ertc_output_type type); |
| Function description   | Set event output, event output on PC13                                                                           |
| Input parameter 1      | source: output source selection<br>Refer to the following "source" descriptions for details.                     |
| Input parameter 2      | polarity: output polarity<br>Refer to the following "polarity" descriptions for details.                         |
| Input parameter3       | type: output type<br>Refer to the following "type" descriptions for details.                                     |
| Output parameter       | NA                                                                                                               |
| Return value           | NA                                                                                                               |
| Required preconditions | NA                                                                                                               |
| Called functions       | NA                                                                                                               |

#### source

Output source selection

- |                      |                      |
|----------------------|----------------------|
| ERTC_OUTPUT_DISABLE: | Output disabled      |
| ERTC_OUTPUT_ALARM_A: | Output alarm A event |
| ERTC_OUTPUT_ALARM_B: | Output alarm B event |
| ERTC_OUTPUT_WAKEUP:  | Output wakeup event  |

#### polarity

Output polarity

- |                            |                                    |
|----------------------------|------------------------------------|
| ERTC_OUTPUT_POLARITY_HIGH: | Output high when an event occurred |
| ERTC_OUTPUT_POLARITY_LOW:  | Output low when an event occurred  |

**type**

Output type

ERTC\_OUTPUT\_TYPE\_OPEN\_DRAIN: Open-drain output

ERTC\_OUTPUT\_TYPE\_PUSH\_PULL: Push-pull output

**Example:**

```
ertc_output_set(ERTC_OUTPUT_ALARM_A, ERTC_OUTPUT_POLARITY_HIGH,
ERTC_OUTPUT_TYPE_PUSH_PULL);
```

### 5.8.36 ertc\_timestamp\_valid\_edge\_set function

The table below describes the function ertc\_timestamp\_valid\_edge\_set.

**Table 190. ertc\_timestamp\_valid\_edge\_set function**

| Name                   | Description                                                                     |
|------------------------|---------------------------------------------------------------------------------|
| Function name          | ertc_timestamp_valid_edge_set                                                   |
| Function prototype     | void ertc_timestamp_valid_edge_set(ertc_timestamp_valid_edge_type edge);        |
| Function description   | Set timestamp detect valid edge                                                 |
| Input parameter 1      | Edge: timestamp detect valid edge<br>Refer to the “edge” below for information. |
| Output parameter       | NA                                                                              |
| Return value           | NA                                                                              |
| Required preconditions | NA                                                                              |
| Called functions       | NA                                                                              |

**edge**

Select a valid edge of timestamp detectin.

ERTC\_TIMESTAMP\_EDGE\_RISING: Rising edge

ERTC\_TIMESTAMP\_EDGE\_FALLING: Falling edge

**Example:**

```
ertc_timestamp_valid_edge_set(ERTC_TIMESTAMP_EDGE_RISING);
```

### 5.8.37 ertc\_timestamp\_enable function

The table below describes the function ertc\_timestamp\_enable.

**Table 191. ertc\_timestamp\_enable function**

| Name                   | Description                                                               |
|------------------------|---------------------------------------------------------------------------|
| Function name          | ertc_timestamp_enable                                                     |
| Function prototype     | void ertc_timestamp_enable(confirm_state new_state);                      |
| Function description   | Enable timestamp                                                          |
| Input parameter 1      | new_state: timestamp enable state<br>This parameter can be TRUE or FALSE. |
| Output parameter       | NA                                                                        |
| Return value           | NA                                                                        |
| Required preconditions | NA                                                                        |
| Called functions       | NA                                                                        |

**Example:**

```
ertc_timestamp_enable(TRUE);
```

## 5.8.38 ertc\_timestamp\_get function

The table below describes the function ertc\_timestamp\_get.

**Table 192. ertc\_timestamp\_get function**

| Name                   | Description                                    |
|------------------------|------------------------------------------------|
| Function name          | ertc_timestamp_get                             |
| Function prototype     | void ertc_timestamp_get(ertc_time_type* time); |
| Function description   | Get timestamp                                  |
| Input parameter 1      | time: ertc_time_type pointer                   |
| Output parameter       | NA                                             |
| Return value           | NA                                             |
| Required preconditions | NA                                             |
| Called functions       | NA                                             |

ertc\_time\_type\* time

The ertc\_time\_type is defined in the “at32f425\_ertc.h”:

typedef struct

```
{
 uint8_t year;
 uint8_t month;
 uint8_t day;
 uint8_t hour;
 uint8_t min;
 uint8_t sec;
 uint8_t week;
 ertc_am_pm_type ampm;
} ertc_time_type;
```

**year**

Range 0~99

**month**

Range 1~12

**day**

Range 1~31

**week**

Range 1~7

**hour**

Range 0~23

**min**

Range 0~59

**sec**

Range 0~59

**ampm**

AM/PM in 12-hour format (only for 12-hour format, doesn't care in 24-hour format), including:

ERTC\_AM: AM in 12-hour format

ERTC\_PM: PM in 12-hour format

**Example:**

```
ertc_timestamp_get(&time);
```

## 5.8.39 ertc\_timestamp\_sub\_second\_get function

The table below describes the function ertc\_timestamp\_sub\_second\_get.

**Table 193. ertc\_timestamp\_sub\_second\_get function**

| Name                   | Description                                   |
|------------------------|-----------------------------------------------|
| Function name          | ertc_timestamp_sub_second_get                 |
| Function prototype     | uint32_t ertc_timestamp_sub_second_get(void); |
| Function description   | Get timestamp subsecond                       |
| Input parameter 1      | NA                                            |
| Output parameter       | NA                                            |
| Return value           | Return timestamp subsecond                    |
| Required preconditions | NA                                            |
| Called functions       | NA                                            |

**Example:**

```
ertc_timestamp_sub_second_get();
```

## 5.8.40 ertc\_tamper\_pull\_up\_enable function

The table below describes the function ertc\_tamper\_pull\_up\_enable.

**Table 194. ertc\_tamper\_pull\_up\_enable function**

| Name                   | Description                                                                                 |
|------------------------|---------------------------------------------------------------------------------------------|
| Function name          | ertc_tamper_pull_up_enable                                                                  |
| Function prototype     | void ertc_tamper_pull_up_enable(confirm_state new_state);                                   |
| Function description   | Enable tamper pin pull-up resistor                                                          |
| Input parameter 1      | new_state: tamper pin pull-up resistor enable state<br>This parameter can be TRUE or FALSE. |
| Output parameter       | NA                                                                                          |
| Return value           | NA                                                                                          |
| Required preconditions | NA                                                                                          |
| Called functions       | NA                                                                                          |

**Example:**

```
ertc_tamper_pull_up_enable(TRUE);
```

## 5.8.41 ertc\_tamper\_purge\_set function

The table below describes the function ertc\_tamper\_purge\_set.

**Table 195. ertc\_tamper\_purge\_set function**

| Name                   | Description                                                                                                                                         |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | ertc_tamper_purge_set                                                                                                                               |
| Function prototype     | void ertc_tamper_purge_set(ertc_tamper_purge_type purge);                                                                                           |
| Function description   | Set tamper pin purge time. This setting is needed only when the tamper pull-up resistor is enabled through the function ertc_tamper_pull_up_enable. |
| Input parameter 1      | purge: tamper pin purge time<br>Refer to the following “purge” descriptions for details.                                                            |
| Output parameter       | NA                                                                                                                                                  |
| Return value           | NA                                                                                                                                                  |
| Required preconditions | NA                                                                                                                                                  |
| Called functions       | NA                                                                                                                                                  |

### purge

Tamper pin purge time

- ERTC\_TAMPER\_PR\_1\_ERTCCLK: One ERTC\_CLK cycle
- ERTC\_TAMPER\_PR\_2\_ERTCCLK: Two ERTC\_CLK cycles
- ERTC\_TAMPER\_PR\_4\_ERTCCLK: Four ERTC\_CLK cycles
- ERTC\_TAMPER\_PR\_8\_ERTCCLK: Eight ERTC\_CLK cycles

### Example:

```
ertc_tamper_purge_set(ERTC_TAMPER_PR_2_ERTCCLK);
```

## 5.8.42 ertc\_tamper\_filter\_set function

The table below describes the function ertc\_tamper\_filter\_set.

**Table 196. ertc\_tamper\_filter\_set function**

| Name                   | Description                                                                                |
|------------------------|--------------------------------------------------------------------------------------------|
| Function name          | ertc_tamper_filter_set                                                                     |
| Function prototype     | void ertc_tamper_filter_set(ertc_tamper_filter_type filter);                               |
| Function description   | Set tamper filtering time                                                                  |
| Input parameter 1      | filter: tamper filtering time<br>Refer to the following “filter” descriptions for details. |
| Output parameter       | NA                                                                                         |
| Return value           | NA                                                                                         |
| Required preconditions | NA                                                                                         |
| Called functions       | NA                                                                                         |

### filter

Set tamper filtering time

ERTC\_TAMPER\_FILTER\_DISABLE:

No filtering

ERTC\_TAMPER\_FILTER\_2:

Tamper event is considered to have occur after two valid consecutive sampling

ERTC\_TAMPER\_FILTER\_4:

Tamper event is considered to have occur after four valid consecutive sampling  
 ERTC\_TAMPER\_FILTER\_8:

Tamper event is considered to have occur after eight valid consecutive sampling

**Example:**

```
ertc_tamper_filter_set(ERTC_TAMPER_FILTER_2);
```

### 5.8.43 ertc\_tamper\_detect\_freq\_set function

The table below describes the function ertc\_tamper\_detect\_freq\_set.

**Table 197. ertc\_tamper\_detect\_freq\_set function**

| Name                   | Description                                                                                 |
|------------------------|---------------------------------------------------------------------------------------------|
| Function name          | ertc_tamper_detect_freq_set                                                                 |
| Function prototype     | void ertc_tamper_detect_freq_set(ertc_tamper_detect_freq_type freq);                        |
| Function description   | Set tamper detection frequency                                                              |
| Input parameter 1      | freq: tamper detection frequency<br>Refer to the following "freq" descriptions for details. |
| Output parameter       | NA                                                                                          |
| Return value           | NA                                                                                          |
| Required preconditions | NA                                                                                          |
| Called functions       | NA                                                                                          |

**freq**

Select tamper detection frequency

|                             |                  |
|-----------------------------|------------------|
| ERTC_TAMPER_FREQ_DIV_32768: | ERTC_CLK / 32768 |
| ERTC_TAMPER_FREQ_DIV_16384: | ERTC_CLK / 16384 |
| ERTC_TAMPER_FREQ_DIV_8192:  | ERTC_CLK / 8192  |
| ERTC_TAMPER_FREQ_DIV_4096:  | ERTC_CLK / 4096  |
| ERTC_TAMPER_FREQ_DIV_2048:  | ERTC_CLK / 2048  |
| ERTC_TAMPER_FREQ_DIV_1024:  | ERTC_CLK / 1024  |
| ERTC_TAMPER_FREQ_DIV_512:   | ERTC_CLK / 512   |
| ERTC_TAMPER_FREQ_DIV_256:   | ERTC_CLK / 256   |

**Example:**

```
ertc_tamper_detect_freq_set(ERTC_TAMPER_FREQ_DIV_512);
```

## 5.8.44 ertc\_tamper\_valid\_edge\_set function

The table below describes the function ertc\_tamper\_valid\_edge\_set.

**Table 198. ertc\_tamper\_valid\_edge\_set function**

| Name                   | Description                                                                                                |
|------------------------|------------------------------------------------------------------------------------------------------------|
| Function name          | ertc_tamper_valid_edge_set                                                                                 |
| Function prototype     | void ertc_tamper_valid_edge_set(ertc_tamper_select_type tamper_x,<br>ertc_tamper_valid_edge_type trigger); |
| Function description   | Set tamper detection valid edge                                                                            |
| Input parameter 1      | tamper_x: tamper selection<br>Refer to the following “tamper_x” descriptions for details.                  |
| Input parameter 2      | trigger: tamper detection valid edge<br>Refer to the following “trigger” descriptions for details.         |
| Output parameter       | NA                                                                                                         |
| Return value           | NA                                                                                                         |
| Required preconditions | NA                                                                                                         |
| Called functions       | NA                                                                                                         |

### **tamper\_x**

Tamper selection

ERTC\_TAMPER\_1: Tamper detection 1

### **trigger**

Tamper detection valid edge selection

ERTC\_TAMPER\_EDGE\_RISING: Rising edge

ERTC\_TAMPER\_EDGE\_FALLING: Falling edge

ERTC\_TAMPER\_EDGE\_LOW: Low level

ERTC\_TAMPER\_EDGE\_HIGH: High level

### **Example:**

```
ertc_tamper_valid_edge_set(ERTC_TAMPER_1, ERTC_TAMPER_EDGE_RISING);
```

## 5.8.45 ertc\_tamper\_timestamp\_enable function

The table below describes the function ertc\_tamper\_timestamp\_enable.

**Table 199. ertc\_tamper\_timestamp\_enable function**

| Name                   | Description                                                                                                    |
|------------------------|----------------------------------------------------------------------------------------------------------------|
| Function name          | ertc_tamper_timestamp_enable                                                                                   |
| Function prototype     | void ertc_tamper_timestamp_enable(confirm_state new_state);                                                    |
| Function description   | Enable timestamp when a tamper event occurred                                                                  |
| Input parameter 1      | new_state: timestamp feature enable state when a tamper event occurred<br>This parameter can be TRUE or FALSE. |
| Output parameter       | NA                                                                                                             |
| Return value           | NA                                                                                                             |
| Required preconditions | NA                                                                                                             |
| Called functions       | NA                                                                                                             |

### **Example:**

```
ertc_tamper_timestamp_enable(TRUE);
```

## 5.8.46 ertc\_tamper\_enable function

The table below describes the function ertc\_tamper\_enable.

**Table 200. ertc\_tamper\_enable function**

| Name                   | Description                                                                               |
|------------------------|-------------------------------------------------------------------------------------------|
| Function name          | ertc_tamper_enable                                                                        |
| Function prototype     | void ertc_tamper_enable(ertc_tamper_select_type tamper_x, confirm_state new_state);       |
| Function description   | Enable tamper detection                                                                   |
| Input parameter 1      | tamper_x: tamper selection<br>Refer to the following “tamper_x” descriptions for details. |
| Input parameter 2      | new_state: tamper detection enable state<br>This parameter can be TRUE or FALSE.          |
| Output parameter       | NA                                                                                        |
| Return value           | NA                                                                                        |
| Required preconditions | NA                                                                                        |
| Called functions       | NA                                                                                        |

### tamper\_x

Tamper selection

ERTC\_TAMPER\_1: Tamper detection 1

### Example:

```
ertc_tamper_enable(ERTC_TAMPER_1, TRUE);
```

## 5.8.47 ertc\_interrupt\_enable function

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

**Table 201. ertc\_interrupt\_enable function**

| Name                   | Description                                                                                         |
|------------------------|-----------------------------------------------------------------------------------------------------|
| Function name          | ertc_interrupt_enable                                                                               |
| Function prototype     | void ertc_interrupt_enable(uint32_t source, confirm_state new_state);                               |
| Function description   | Interrupt enable                                                                                    |
|                        | source: interrupt source to be enabled<br>Refer to the following “source” descriptions for details. |
| Input parameter 1      | new_state: interrupt enable state<br>This parameter can be TRUE or FALSE.                           |
| Output parameter       | NA                                                                                                  |
| Return value           | NA                                                                                                  |
| Required preconditions | NA                                                                                                  |
| Called functions       | NA                                                                                                  |

### source

Interrupt source to be enabled

ERTC\_TP\_INT: Tamper detection interrupt

ERTC\_ALA\_INT: Alarm A interrupt

ERTC\_WAT\_INT: Wakeup timer interrupt

ERTC\_TS\_INT: Time stamp interrupt

**Example:**

```
ertc_interrupt_enable(ERTC_TP_INT, TRUE);
```

### 5.8.48 ertc\_interrupt\_get function

The table below describes the function ertc\_interrupt\_get.

**Table 202. ertc\_interrupt\_get function**

| Name                   | Description                                                                           |
|------------------------|---------------------------------------------------------------------------------------|
| Function name          | ertc_interrupt_get                                                                    |
| Function prototype     | flag_status ertc_interrupt_get(uint32_t source);                                      |
| Function description   | Get interrupt enable state                                                            |
| Input parameter 1      | source: interrupt source<br>Refer to the following "source" descriptions for details. |
| Output parameter       | NA                                                                                    |
| Return value           | flag_status: flag status<br>This parameter can be SET or RESET.                       |
| Required preconditions | NA                                                                                    |
| Called functions       | NA                                                                                    |

**source**

Interrupt source

ERTC\_TP\_INT: Tamper detection interrupt

ERTC\_ALA\_INT: Alarm A interrupt

ERTC\_WAT\_INT: Wakeup timer interrupt

ERTC\_TS\_INT: Time stamp interrupt

**Example:**

```
ertc_interrupt_get(ERTC_TP_INT);
```

### 5.8.49 ertc\_flag\_get function

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

**Table 203. ertc\_flag\_get function**

| Name                   | Description                                                                     |
|------------------------|---------------------------------------------------------------------------------|
| Function name          | ertc_flag_get                                                                   |
| Function prototype     | flag_status ertc_flag_get(uint32_t flag);                                       |
| Function description   | Get flag status                                                                 |
| Input parameter 1      | flag: flag selection<br>Refer to the following "flag" descriptions for details. |
| Output parameter       | NA                                                                              |
| Return value           | flag_status: flag status<br>This parameter can be SET or RESET.                 |
| Required preconditions | NA                                                                              |
| Called functions       | NA                                                                              |

**flag**

This bit is used to select a flag. Optional parameters are as follows:

ERTC\_ALAWF\_FLAG: Alarm A write enable flag

|                    |                                         |
|--------------------|-----------------------------------------|
| ERTC_WATWF_FLAG:   | Wakeup timer register write enable flag |
| ERTC_TADJF_FLAG:   | Time adjust flag                        |
| ERTC_INITF_FLAG:   | Calendar initialization flag            |
| ERTC_UPDF_FLAG:    | Calendar update flag                    |
| ERTC_IMF_FLAG:     | Initialization mode entry flag          |
| ERTC_ALAF_FLAG:    | Alarm A flag                            |
| ERTC_WATF_FLAG:    | Wakeup timer flag                       |
| ERTC_TSFLAG:       | Time stamp flag                         |
| ERTC_TSOF_FLAG:    | Time stamp overflow flag                |
| ERTC_TP1F_FLAG:    | Tamper detection 1 flag                 |
| ERTC_CALUPDF_FLAG: | Calibration value update complete flag  |

**Example:**

```
ertc_flag_get(ERTC_TP1F_FLAG);
```

### 5.8.50 ertc\_interrupt\_flag\_get function

The table below describes the function ertc\_interrupt\_flag\_get

**Table 204. ertc\_interrupt\_flag\_get function**

| Name                   | Description                                                                 |
|------------------------|-----------------------------------------------------------------------------|
| Function name          | ertc_interrupt_flag_get                                                     |
| Function prototype     | flag_status ertc_interrupt_flag_get(uint32_t flag);                         |
| Function description   | Get ERTC interrupt flag status and check corresponding interrupt enable bit |
| Input parameter 1      | flag: flag selection<br>Refer to the "flag" below for details.              |
| Output parameter       | NA                                                                          |
| Return value           | flag_status: SET or RESET                                                   |
| Required preconditions | NA                                                                          |
| Called functions       | NA                                                                          |

**flag**

This bit is used to select a flag. Optional parameters are as follows:

|                 |                         |
|-----------------|-------------------------|
| ERTC_ALAF_FLAG: | Alarm A flag            |
| ERTC_WATF_FLAG: | Wakeup timer flag       |
| ERTC_TSFLAG:    | Time stamp flag         |
| ERTC_TP1F_FLAG: | Tamper detection 1 flag |

**Example:**

```
ertc_interrupt_flag_get(ERTC_TP1F_FLAG);
```

### 5.8.51 ertc\_flag\_clear function

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

**Table 205. ertc\_flag\_clear function**

| Name                 | Description                          |
|----------------------|--------------------------------------|
| Function name        | ertc_flag_clear                      |
| Function prototype   | void ertc_flag_clear(uint32_t flag); |
| Function description | Clear flag                           |

| Name                   | Description                                                                     |
|------------------------|---------------------------------------------------------------------------------|
| Input parameter 1      | flag: flag selection<br>Refer to the following "flag" descriptions for details. |
| Output parameter       | NA                                                                              |
| Return value           | NA                                                                              |
| Required preconditions | NA                                                                              |
| Called functions       | NA                                                                              |

**flag**

This bit is used to select a flag. Optional parameters are as follows:

|                    |                                         |
|--------------------|-----------------------------------------|
| ERTC_ALAWF_FLAG:   | Alarm A write enable flag               |
| ERTC_WATWF_FLAG:   | Wakeup timer register write enable flag |
| ERTC_TADJF_FLAG:   | Time adjust flag                        |
| ERTC_INITF_FLAG:   | Calendar initialization flag            |
| ERTC_UPDF_FLAG:    | Calendar update flag                    |
| ERTC_IMF_FLAG:     | Initialization mode entry flag          |
| ERTC_ALAF_FLAG:    | Alarm A flag                            |
| ERTC_WATF_FLAG:    | Wakeup timer flag                       |
| ERTC_TSOF_FLAG:    | Time stamp flag                         |
| ERTC_TSOF_FLAG:    | Time stamp overflow flag                |
| ERTC_TP1F_FLAG:    | Tamper detection 1 flag                 |
| ERTC_CALUPDF_FLAG: | Calibration value update complete flag  |

**Example:**

```
ertc_flag_clear(ERTC_TP1F_FLAG);
```

### 5.8.52 ertc\_bpr\_data\_write function

The table below describes the function ertc\_bpr\_data\_write.

**Table 206. ertc\_bpr\_data\_write function**

| Name                   | Description                                                                |
|------------------------|----------------------------------------------------------------------------|
| Function name          | ertc_bpr_data_write                                                        |
| Function prototype     | void ertc_bpr_data_write(ertc_dt_type dt, uint32_t data);                  |
| Function description   | Write data to BPR register (battery powered data register)                 |
| Input parameter 1      | dt: data register<br>Refer to the following "dt" descriptions for details. |
| Input parameter 1      | data: 32-bit data                                                          |
| Output parameter       | NA                                                                         |
| Return value           | NA                                                                         |
| Required preconditions | NA                                                                         |
| Called functions       | NA                                                                         |

**dt**

Data register

ERTC\_DT1: Data register 1

ERTC\_DT2: Data register 2

ERTC\_DT3: Data register 3

ERTC\_DT4: Data register 4

ERTC\_DT5: Data register 5

**Example:**

```
ertc_bpr_data_write(ERTC_DT1, 0x12345678);
```

## 5.8.53 ertc\_bpr\_data\_read function

The table below describes the function ertc\_bpr\_data\_read.

**Table 207. ertc\_bpr\_data\_read function**

| Name                   | Description                                                                |
|------------------------|----------------------------------------------------------------------------|
| Function name          | ertc_bpr_data_read                                                         |
| Function prototype     | uint32_t ertc_bpr_data_read(ertc_dt_type dt);                              |
| Function description   | Read data from BPR register (battery powered data register)                |
| Input parameter 1      | dt: data register<br>Refer to the following "dt" descriptions for details. |
| Output parameter       | NA                                                                         |
| Return value           | Data from BPR register                                                     |
| Required preconditions | NA                                                                         |
| Called functions       | NA                                                                         |

**dt**

Data register

ERTC\_DT1: Data register 1

ERTC\_DT2: Data register 2

ERTC\_DT3: Data register 3

ERTC\_DT4: Data register 4

ERTC\_DT5: Data register 5

### Example:

```
ertc_bpr_data_read(ERTC_DT1);
```

## 5.9 External interrupt/event controller (EXINT)

The EXINT register structure exint\_type is defined in the “at32f425\_exint.h”:

```
/*
 * @brief type define exint register all
 */
typedef struct
{
 ...
} exint_type;
```

The table below gives a list of the EXINT registers:

**Table 208. 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 209. 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 flag                                 |
| exint_interrupt_flag_get                | Get the selected EXINT interrupt flag status                 |
| 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 210. 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 211. 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 212. 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      | <a href="#">exint_init_type</a> : exint_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                                                                       |

The exint\_init\_type is defined in the “at32f425\_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: None

EXINT\_LINE\_0: line0

EXINT\_LINE\_1: line1

...

EXINT\_LINE\_19: line19

EXINT\_LINE\_20: line20

#### 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 213. 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<br>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 214. 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<br>Refer to <a href="#">line_select</a> for details.               |
| Output parameter       | NA                                                                                            |
| Return value           | flag_status: indicates the status of the selected flag<br>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 215. 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 EXINT interrupt flag status                                                 |
| Input parameter        | exint_line: line selection<br>Refer to <a href="#">line_select</a> for details. |
| Output parameter       | NA                                                                              |
| Return value           | flag_status: 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 216. 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<br>Refer to <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 217. 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<br>Refer to <a href="#">line_select</a> for details. |
| Input parameter 2      | new_state: Enable or disable<br>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 218. 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<br>Refer to <a href="#">line_select</a> for details. |
| Input parameter 2      | new_state: Enable or disable<br>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 “at32f425\_flash.h”:

```
/**
 * @brief type define flash register all
 */
typedef struct
{
 ...
} flash_type;
```

The table below gives a list of the FLASH registers

Table 219. 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          |
| slib_sts0        | Flash security library status register 0          |
| slib_sts1        | Flash security library status register 1          |
| slib_pwd_clr     | Flash security library password clear register    |
| slib_misc_sts    | Flash security library extra status register      |
| Flash_crc_addr   | Flash CRC address register                        |
| flash_crc_ctrl   | Flash CRC check control register                  |
| flash_crc_chkr   | Flash CRC check result register                   |
| slib_set_pwd     | Flash security library password setting register  |
| slib_set_range   | Flash security library address setting register   |
| em_slib_set      | Extended memory security library setting register |
| btm_mode_set     | Boot memory mode setting register                 |
| slib_unlock      | Flash security library unlock register            |

The table below gives a list of FLASH library functions.

**Table 220. Summary of FLASH library functions**

| Function name                           | Description                                                                       |
|-----------------------------------------|-----------------------------------------------------------------------------------|
| flash_flag_get                          | Get flag status                                                                   |
| flash_flag_clear                        | Clear flag                                                                        |
| flash_operation_status_get              | Get Flash operation status (Flash memory bank 1)                                  |
| flash_operation_wait_for                | Wait for operation complete (Flash memory bank 1)                                 |
| flash_unlock                            | Unlock Flash (Flash memory bank 1 and 2)                                          |
| flash_lock                              | Lock Flash (Flash memory bank 1 and 2)                                            |
| flash_sector_erase                      | Erase Flash sector                                                                |
| flash_internal_all_erase                | Erase internal Flash                                                              |
| 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_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_fap_high_level_enable             | Flash high level access protection enable                                         |
| flash_fap_high_level_status_get         | Get Flash high level 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_slib_enable                       | sLib enable                                                                       |
| flash_slib_disable                      | sLib disable                                                                      |
| flash_slib_state_get                    | Get sLib states                                                                   |
| flash_slib_start_sector_get             | Get sLib start sector                                                             |
| flash_slib_datastart_sector_get         | Get sLib data area start sector                                                   |
| flash_slib_end_sector_get               | Get sLib end sector                                                               |
| flash_crc_calibrate                     | Flash CRC verify                                                                  |
| flash_boot_memory_extension_mode_enable | Boot memory is used as an extended Flash memory                                   |
| flash_extension_memory_slib_enable      | Extended Flash memory is used as a security library                               |
| flash_extension_memory_slib_state_get   | Get the status of extended Flash memory which is used as a security library       |
| flash_em_slib_inststart_sector_get      | Get the start page of instruction area of security library in the extended memory |
| flash_low_power_mode_enable             | Enable Flash low-power mode                                                       |

## 5.10.1 flash\_flag\_get function

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

Table 221. 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<br>Return RESET or SET |
| Required preconditions | NA                                                            |
| Called functions       | NA                                                            |

### flash\_flag

Flash status flag selection.

|                    |                             |
|--------------------|-----------------------------|
| FLASH_OBF_FLAG:    | Flash operation busy        |
| FLASH_ODF_FLAG:    | Flash operation complete    |
| FLASH_PGMERR_FLAG: | Flash programming error     |
| FLASH_EPPERR_FLAG: | Flash 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 222. 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

Flash status flag selection.

FLASH\_ODF\_FLAG: Flash operation complete

FLASH\_PRGMERR\_FLAG: Flash programming error

FLASH\_EPPERR\_FLAG: Flash 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 223. 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\_operation\_wait\_for function

The table below describes the function flash\_operation\_wait\_for.

**Table 224. 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<br>The timeout value is defined in the flash.h file. |
| 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: Program timeout

OPERATION\_TIMEOUT: Other general operation timeout

### Example:

```
/* wait for operation to be completed */
status = flash_operation_wait_for(PROGRAMMING_TIMEOUT);
```

## 5.10.5 flash\_unlock function

The table below describes the function flash\_unlock.

**Table 225. 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();
```

## 5.10.6 flash\_lock function

The table below describes the function flash\_lock.

**Table 226. 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();
```

## 5.10.7 flash\_sector\_erase function

The table below describes the function flash\_sector\_erase.

**Table 227. 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);
```

## 5.10.8 flash\_internal\_all\_erase function

The table below describes the function flash\_internal\_all\_erase.

**Table 228. 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. |

| Name                   | Description |
|------------------------|-------------|
| Required preconditions | NA          |
| Called functions       | NA          |

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_internal_all_erase();
```

### 5.10.9 flash\_user\_system\_data\_erase function

The table below describes the function `flash_user_system_data_erase`.

**Table 229. `flash_user_system_data_erase` function**

| Name                   | Description                                                        |
|------------------------|--------------------------------------------------------------------|
| Function name          | <code>flash_user_system_data_erase</code>                          |
| Function prototype     | <code>flash_status_type flash_user_system_data_erase(void);</code> |
| 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();
```

### 5.10.10 flash\_word\_program function

The table below describes the function `flash_word_program`.

**Table 230. `flash_word_program` function**

| Name                   | Description                                                                         |
|------------------------|-------------------------------------------------------------------------------------|
| Function name          | <code>flash_word_program</code>                                                     |
| Function prototype     | <code>flash_status_type flash_word_program(uint32_t address, uint32_t data);</code> |
| Function description   | Write one word data to a given address                                              |
| Input parameter 1      | Address: programmed address, word-aligned                                           |
| 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 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);
 }
}
```

### 5.10.11 flash\_halfword\_program function

The table below describes the function flash\_halfword\_program.

**Table 231. 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   | Write a half-word data to a given address                                           |
| Input parameter 1      | Address: programmed address, half-word-aligned                                      |
| 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 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);
 }
}
```

## 5.10.12 flash\_byte\_program function

The table below describes the function flash\_byte\_program.

**Table 232. 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: 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 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);
 }
}
```

## 5.10.13 flash\_user\_system\_data\_program function

The table below describes the function flash\_user\_system\_data\_program.

**Table 233. 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(0x1FFF804, 0x55);
}

```

**5.10.14 flash\_epp\_set function**

The table below describes the function `flash_epp_set`.

**Table 234. `flash_epp_set` function**

| Name                   | Description                                                                                                                                                                                                   |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | <code>flash_epp_set</code>                                                                                                                                                                                    |
| Function prototype     | <code>flash_status_type flash_epp_set(uint32_t *sector_bits);</code>                                                                                                                                          |
| Function description   | Enable erase programming protection                                                                                                                                                                           |
| Input parameter        | *sector_bits:<br>Erase programming protection sector address pointer. Each bit in bits 15~0<br>protects 4KB sectors, bit 31 protects Flash extended area. Setting this bit to 1<br>enables sector protection. |
| Output parameter       | NA                                                                                                                                                                                                            |
| Return value           | Return operation status. 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[2];
flash_unlock();
status = flash_user_system_data_erase();
if(status = FLASH_OPERATE_DONE)
{
 epp_val[0] = 0x00000001;
 epp_val[1] = 0x00000001;
 /* program epp */
 status = flash_epp_set(epp_val);
}

```

## 5.10.15 flash\_epp\_status\_get function

The table below describes the function flash\_epp\_status\_get.

**Table 235. 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:<br>Erase programming protection sector address pointer. Each bit in bits 15~0 protects 4KB sectors, bit 31 protects Flash extended area. Setting this bit to 1 enables sector protection. |
| Return value           | NA                                                                                                                                                                                                      |
| Required preconditions | NA                                                                                                                                                                                                      |
| Called functions       | NA                                                                                                                                                                                                      |

**Example:**

```
uint32_t epp_val[2];
/* get epp status */
flash_epp_status_get(epp_val);
```

## 5.10.16 flash\_fap\_enable function

The table below describes the function flash\_fap\_enable.

**Table 236. 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<br>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);
```

### 5.10.17 flash\_fap\_status\_get function

The table below describes the function flash\_fap\_status\_get.

**Table 237. 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<br>This parameter can be SET or RESET. |
| Required preconditions | NA                                                              |
| Called functions       | NA                                                              |

**Example:**

```
flag_status status;
status = flash_fap_status_get();
```

### 5.10.18 flash\_fap\_high\_level\_enable

The table below describes the function flash\_fap\_high\_level\_enable.

**Table 238. flash\_fap\_high\_level\_enable function**

| Name                   | Description                                                                                                              |
|------------------------|--------------------------------------------------------------------------------------------------------------------------|
| Function name          | flash_fap_high_level_enable                                                                                              |
| Function prototype     | flash_status_type flash_fap_high_level_enable (void);                                                                    |
| Function description   | Enable Flash high level access protection                                                                                |
| Input parameter        | NA                                                                                                                       |
| Output parameter       | NA                                                                                                                       |
| Return value           | flag_status: flag status. Refer to <a href="#">flash_status_type</a> for details.<br>This parameter can be SET or RESET. |
| Required preconditions | NA                                                                                                                       |
| Called functions       | NA                                                                                                                       |

*Note: Once the high-level access protection is enabled, SWD debug interface will be permanently forbidden.*

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
flash_unlock();
status = flash_fap_high_level_enable();
```

### 5.10.19 flash\_fap\_high\_level\_status\_get

The table below describes the function flash\_fap\_high\_level\_status\_get.

**Table 239. flash\_fap\_high\_level\_status\_get function**

| Name                   | Description                                                     |
|------------------------|-----------------------------------------------------------------|
| Function name          | flash_fap_high_level_status_get                                 |
| Function prototype     | flash_status_flash_fap_high_level_status_get (void);            |
| Function description   | Get Flash high level access protection status                   |
| Input parameter        | NA                                                              |
| Output parameter       | NA                                                              |
| Return value           | flag_status: flag status<br>This parameter can be SET or RESET. |
| Required preconditions | NA                                                              |
| Called functions       | NA                                                              |

**Example:**

```
flag_status status;
status = flash_fap_high_level_status_get();
```

### 5.10.20 flash(ssb)\_set function

The table below describes the function flash(ssb)\_set.

**Table 240. flash(ssb)\_set function**

| Name                   | Description                                                                                                                                            |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | flash_ss�_set                                                                                                                                          |
| Function prototype     | flash_status_type flash_ss�_set(uint8_t usd_ss�);                                                                                                      |
| Function description   | Configure system setting bytes                                                                                                                         |
| Input parameter        | usd_ss�:<br>system setting byte value is a combination of the selected data from all data group, refer to <a href="#">ss�_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                                                                                                                                                     |

**ss�\_data\_define**

USD\_WDT\_ATO\_DISABLE: Watchdog auto-start disabled

USD\_WDT\_ATO\_ENABLE: Watchdog auto-start enabled

USD\_DEPSLP\_NO\_RST: No reset occurred when entering Deepsleep mode

USD\_DEPSLP\_RST: Reset occurred when entering Deepsleep mode

USD\_STDBY\_NO\_RST: No reset occurred when entering Standby mode

USD\_STDBY\_RST: Reset occurred when entering Standby mode

USD\_BOOT1\_LOW: When BOOT0 is low, Flash starts from boot memory

USD\_BOOT1\_HIGH: When BOOT0 high, Flash starts from internal memory

USD\_WDT\_DEPSLP\_CONTINUE: WDT continues counting in Deepsleep mode  
 USD\_WDT\_DEPSLP\_STOP: WDT stops counting in Deepsleep mode

USD\_WDT\_STDBY\_CONTINUE: WDT continues counting in Standby mode  
 USD\_WDT\_STDBY\_STOP: WDT stops counting in Standby mode

**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);
}
```

### 5.10.21 flash\_ssb\_status\_get function

The table below describes the function flash\_ssb\_status\_get.

**Table 241. flash\_ssb\_status\_get function**

| Name                   | Description                                                                             |
|------------------------|-----------------------------------------------------------------------------------------|
| Function name          | flash_ssb_status_get                                                                    |
| Function prototype     | uint8_t flash_ssb_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_ssb_status_get();
```

### 5.10.22 flash\_interrupt\_enable function

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

**Table 242. 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<br>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 |

**Example:**

```
flash_interrupt_enable(FLASH_BANK1_ERR_INT | FLASH_BANK1_ODF_INT, TRUE);
```

### 5.10.23 flash\_slib\_enable function

The table below describes the function `flash_slib_enable`.

**Table 243. flash\_slib\_enable function**

| Name                   | Description                                                                                                                                                  |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | <code>flash_slib_enable</code>                                                                                                                               |
| Function prototype     | <code>flash_status_type flash_slib_enable(uint32_t pwd, uint16_t start_sector, uint16_t inst_start_sector, uint16_t end_sector);</code>                      |
| Function description   | Enable security library (sLib) and its address range                                                                                                         |
| Input parameter 1      | Pwd: sLib password. The sLib data are saved as ciphertext, associated with encrypted computing. A correct password is entered in order to unlock encryption. |
| Input parameter 2      | start_sector: sLib start sector number                                                                                                                       |
| Input parameter 3      | inst_start_sector: sLib data area instruction start sector number                                                                                            |
| Input parameter 4      | end_sector: sLib end sector number                                                                                                                           |
| 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;
status = flash_slib_enable(0x12345678, 0x04, 0x05, 0x06);
```

### 5.10.24 flash\_slib\_disable function

The table below describes the function `flash_slib_disable`.

**Table 244. flash\_slib\_disable function**

| Name                   | Description                                                                                         |
|------------------------|-----------------------------------------------------------------------------------------------------|
| Function name          | <code>flash_slib_disable</code>                                                                     |
| Function prototype     | <code>error_status flash_slib_disable(uint32_t pwd);</code>                                         |
| Function description   | Disable security library (sLib)                                                                     |
| Input parameter        | Pwd: sLib password. it must be entered correctly, otherwise it is not allowed to enter until reset. |
| Output parameter       | NA                                                                                                  |
| Return value           | Return error status<br>This parameter can be ERROE or SUCCESS.                                      |
| Required preconditions | NA                                                                                                  |
| Called functions       | NA                                                                                                  |

*Note: Successful calling of this function will erase the whole internal Flash memory.*

**Example:**

```
error_status status;
status = flash_slib_disable(0x12345678);
```

## 5.10.25 flash\_slib\_state\_get function

The table below describes the function flash\_slib\_state\_get.

**Table 245. flash\_slib\_state\_get function**

| Name                   | Description                                                     |
|------------------------|-----------------------------------------------------------------|
| Function name          | flash_slib_state_get                                            |
| Function prototype     | flag_status flash_slib_state_get(void);                         |
| Function description   | Get the status of sLib                                          |
| Input parameter        | NA                                                              |
| Output parameter       | NA                                                              |
| Return value           | flag_status: flag status<br>This parameter can be SET or RESET. |
| Required preconditions | NA                                                              |
| Called functions       | NA                                                              |

**Example:**

```
flag_status status;
status = flash_slib_state_get();
```

## 5.10.26 flash\_slib\_start\_sector\_get function

The table below describes the function flash\_slib\_start\_sector\_get.

**Table 246. flash\_slib\_start\_sector\_get function**

| Name                   | Description                                 |
|------------------------|---------------------------------------------|
| Function name          | flash_slib_start_sector_get                 |
| Function prototype     | uint16_t flash_slib_start_sector_get(void); |
| Function description   | Get the start sector number of sLib         |
| Input parameter        | NA                                          |
| Output parameter       | NA                                          |
| Return value           | Return the start sector number of sLib      |
| Required preconditions | NA                                          |
| Called functions       | NA                                          |

**Example:**

```
uint16_t num;
num = flash_slib_start_sector_get();
```

### 5.10.27 flash\_slib\_inststart\_sector\_get function

The table below describes the function flash\_slib\_inststart\_sector\_get.

**Table 247. flash\_slib\_inststart\_sector\_get function**

| Name                   | Description                                             |
|------------------------|---------------------------------------------------------|
| Function name          | flash_slib_inststart_sector_get                         |
| Function prototype     | uint16_t flash_slib_inststart_sector_get(void);         |
| Function description   | Get the start sector number of sLib instruction area    |
| Input parameter        | NA                                                      |
| Output parameter       | NA                                                      |
| Return value           | Return the start sector number of sLib instruction area |
| Required preconditions | NA                                                      |
| Called functions       | NA                                                      |

**Example:**

```
uint16_t num;
num = flash_slib_inststart_sector_get();
```

### 5.10.28 flash\_slib\_end\_sector\_get function

The table below describes the function flash\_slib\_end\_sector\_get.

**Table 248. flash\_slib\_end\_sector\_get function**

| Name                   | Description                               |
|------------------------|-------------------------------------------|
| Function name          | flash_slib_end_sector_get                 |
| Function prototype     | uint16_t flash_slib_end_sector_get(void); |
| Function description   | Get the end sector number of sLib         |
| Input parameter        | NA                                        |
| Output parameter       | NA                                        |
| Return value           | Return the end sector number of sLib      |
| Required preconditions | NA                                        |
| Called functions       | NA                                        |

**Example:**

```
uint16_t num;
num = flash_slib_end_sector_get();
```

### 5.10.29 flash\_crc\_calibrate function

The table below describes the function flash\_crc\_calibrate.

**Table 249. flash\_crc\_calibrate function**

| Name                 | Description                                                               |
|----------------------|---------------------------------------------------------------------------|
| Function name        | flash_crc_calibrate                                                       |
| Function prototype   | uint32_t flash_crc_calibrate(uint32_t start_sector, uint32_t sector_cnt); |
| Function description | Enable Flash CRC check                                                    |
| Input parameter 1    | start_addr: CRC check start address                                       |
| Input parameter 2    | sector_cnt: CRC check sector count                                        |
| Output parameter     | NA                                                                        |

| Name                   | Description                   |
|------------------------|-------------------------------|
| Return value           | Return CRC calculation result |
| Required preconditions | NA                            |
| Called functions       | NA                            |

*Note: The sector set to go through CRC check is only allowed to be on a single area, rather than on both security library and common area.*

**Example:**

```
uint32_t crc_val;
crc_val = flash_crc_calibrate(0, 10);
```

### 5.10.30 flash\_boot\_memory\_extension\_mode\_enable

The table describes the flash\_boot\_memory\_extension\_mode\_enable

**Table 250. flash\_boot\_memory\_extension\_mode\_enable**

| Name                   | Description                                                                                         |
|------------------------|-----------------------------------------------------------------------------------------------------|
| Function name          | flash_boot_memory_extension_mode_enable                                                             |
| Function prototype     | void flash_boot_memory_extension_mode_enable (void);                                                |
| Function description   | Boot memory is used as extended Flash memory<br>This configuration takes effect after system reset. |
| Input parameter        | NA                                                                                                  |
| Output parameter       | NA                                                                                                  |
| Return value           | NA                                                                                                  |
| Required preconditions | NA                                                                                                  |
| Called functions       | NA                                                                                                  |

*Note: This feature is irreversible, meaning that once it becomes effective, it is no longer possible to resume boot memory.*

**Example:**

```
flash_boot_memory_extension_mode_enable();
nvic_system_reset();
```

### 5.10.31 flash\_extension\_memory\_slib\_enable

The table describes the flash\_extension\_memory\_slib\_enable

**Table 251. flash\_extension\_memory\_slib\_enable**

| Name                 | Description                                                                                                                                                  |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name        | flash_extension_memory_slib_enable                                                                                                                           |
| Function prototype   | flash_status_type flash_extension_memory_slib_enable(uint32_t pwd, uint16_t inst_start_sector);                                                              |
| Function description | Enable extended Flash memory as security library and set the start sector of the security library data area                                                  |
| Input parameter 1    | Pwd: sLib password. In security library, data are stored in the form of cipher. Thus a correct password is required in order to unlock the security library. |
| Input parameter 2    | inst_start_sector: the start sector of security library instruction area when extended Flash memory is used as security library                              |
| Output parameter     | NA                                                                                                                                                           |
| Return value         | Operature status, see <a href="#">flash_status_type</a>                                                                                                      |

| Name                   | Description |
|------------------------|-------------|
| Required preconditions | NA          |
| Called functions       | NA          |

Note: Only either Flash memory or its extended area can be configured as security library at a time, meaning that such two areas cannot be used as security library at the same time.

**Example:**

```
flash_status_type status = FLASH_OPERATE_DONE;
status = flash_extension_memory_slib_enable(0x123456, 0x01);
```

### 5.10.32 flash\_extension\_memory\_slib\_state\_get

The table describes the flash\_extension\_memory\_slib\_state\_get

**Table 252. flash\_extension\_memory\_slib\_state\_get**

| Name                   | Description                                                         |
|------------------------|---------------------------------------------------------------------|
| Function name          | flash_extension_memory_slib_state_get                               |
| Function prototype     | flag_status flash_extension_memory_slib_state_get (void);           |
| Function description   | Get the status of the security library in the extended Flash memory |
| Input parameter        | NA                                                                  |
| Output parameter       | NA                                                                  |
| Return value           | flag_status: flag status<br>This value can be SET or RESET.         |
| Required preconditions | NA                                                                  |
| Called functions       | NA                                                                  |

**Example:**

```
flag_status status;
status = flash_extension_memory_slib_state_get();
```

### 5.10.33 flash\_em\_slib\_inststart\_sector\_get

The table describes the flash\_em\_slib\_inststart\_sector\_get

**Table 253. flash\_em\_slib\_inststart\_sector\_get**

| Name                   | Description                                                                               |
|------------------------|-------------------------------------------------------------------------------------------|
| Function name          | flash_em_slib_inststart_sector_get                                                        |
| Function prototype     | uint16_t flash_em_slib_inststart_sector_get (void);                                       |
| Function description   | Get the start sector of security library instruction area in the extended Flash memory    |
| Input parameter        | NA                                                                                        |
| Output parameter       | NA                                                                                        |
| Return value           | Return the start sector of security library instruction area in the extended Flash memory |
| Required preconditions | NA                                                                                        |
| Called functions       | NA                                                                                        |

**Example:**

```
uint16_t num;
num = flash_em_slib_datastart_sector_get();
```

### 5.10.34 flash\_low\_power\_mode\_enable

The table describes the flash\_low\_power\_mode\_enable function.

**Table 254. flash\_low\_power\_mode\_enable**

| Name                   | Description                                                        |
|------------------------|--------------------------------------------------------------------|
| Function name          | flash_low_power_mode_enable                                        |
| Function prototype     | void flash_low_power_mode_enable(confirm_state new_state);         |
| Function description   | Enable Flash low-power mode                                        |
| Input parameter        | new_state: low-power state<br>This parameter can be TRUE or FALSE. |
| Output parameter       | NA                                                                 |
| Return value           | NA                                                                 |
| Required preconditions | NA                                                                 |
| Called functions       | NA                                                                 |

**Example:**

```
flash_low_power_mode_enable(TRUE);
```

## 5.11 General-purpose I/Os and multiplexed I/Os (GPIO/IOMUX)

The GPIO register structure gpio\_type is defined in the “at32f425\_gpio.h”:

```
/*
 * @brief type define gpio register all
 */
typedef struct
{
} gpio_type;
```

Table 299 gives a list of the GPIO registers

**Table 255. Summary of GPIO registers**

| Register | Description                                   |
|----------|-----------------------------------------------|
| cfg      | GPIO configuration register                   |
| omode    | GPIO output mode register                     |
| odrvr    | GPIO drive capability switch control register |
| pull     | GPIO pull-up/pull-down register               |
| idt      | GPIO input register                           |
| odt      | GPIO output register                          |
| scr      | GPIO set/clear register                       |
| wpr      | GPIO write protection register                |
| muxl     | GPIO multiplexed function low register        |
| muxh     | GPIO multiplexed function high register       |
| clr      | GPIO port bit clear register                  |
| hdrv     | GPIO huge current control register            |

Table 300 gives a list of GPIO and IOMUX library functions.

**Table 256. GPIO and IOMUX library functions**

| Function name                | Description                             |
|------------------------------|-----------------------------------------|
| gpio_reset                   | GPIO 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_pins_huge_driven_config | Configure GPIO huge drive capability    |
| gpio_pin_mux_config          | Configure GPIO pin multiplexed function |

## 5.11.1 gpio\_reset function

The table below describes the function gpio\_reset.

**Table 257. 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 a GPIO peripheral.<br>GPIOA, GPIOB, GPIOC, GPIOD, GPIOF |
| Output parameter       | NA                                                                     |
| Return value           | NA                                                                     |
| Required preconditions | NA                                                                     |
| Called functions       | crm_periph_reset();                                                    |

**Example:**

```
gpio_reset(GPIOA);
```

## 5.11.2 gpio\_init function

The table below describes the function gpio\_init.

**Table 258. 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: the selected GPIO peripheral<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF. |
| 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 at32f425\_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 mode       |
| GPIO_MODE_OUTPUT: | GPIO output mode      |
| 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);
```

### 5.11.3 gpio\_default\_para\_init function

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

**Table 259. 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 parameters                             |
| 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 260. 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);
```

### 5.11.4 gpio\_input\_data\_bit\_read function

The table below describes the function gpio\_input\_data\_bit\_read.

**Table 261. 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 pins                                                                          |
| Input parameter 1      | gpio_x: the selected GPIO peripheral<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF. |
| 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);
```

## 5.11.5 gpio\_input\_data\_read function

The table below describes the function gpio\_input\_data\_read.

**Table 262. 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 ports                                                                                         |
| Input parameter        | gpio_x: indicates the selected GPIO peripheral.<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF. |
| Output parameter       | NA                                                                                                            |
| Return value           | Return GPIO input port status                                                                                 |
| Required preconditions | NA                                                                                                            |
| Called functions       | NA                                                                                                            |

**Example:**

```
gpio_input_data_read(GPIOA);
```

## 5.11.6 gpio\_output\_data\_bit\_read function

The table below describes the function gpio\_output\_data\_bit\_read.

**Table 263. 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<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF. |
| 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);
```

## 5.11.7 gpio\_output\_data\_read function

The table below describes the function gpio\_output\_data\_read.

**Table 264. 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: the selected GPIO peripheral<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF. |
| Output parameter     | NA                                                                                                 |

| Name                   | Description                  |
|------------------------|------------------------------|
| Return value           | Read GPIO output port status |
| Required preconditions | NA                           |
| Called functions       | NA                           |

**Example:**

```
gpio_output_data_read(GPIOA);
```

### 5.11.8 gpio\_bits\_set function

The table below describes the function gpio\_bits\_set.

**Table 265. 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<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD, GPIOE or GPIOF. |
| 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);
```

### 5.11.9 gpio\_bits\_reset function

The table below describes the function gpio\_bits\_reset.

**Table 266. 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<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF. |
| 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);
```

## 5.11.10 gpio\_bits\_write function

The table below describes the function gpio\_bits\_write.

**Table 267. gpio\_bits\_write function**

| Name                   | Description                                                                                                  |
|------------------------|--------------------------------------------------------------------------------------------------------------|
| Function name          | gpio_bits_toggle                                                                                             |
| Function prototype     | void gpio_bits_toggle (gpio_type *gpio_x, uint16_t pins);                                                    |
| Function description   | Write GPIO pins                                                                                              |
| Input parameter 1      | gpio_x: indicates the selected GPIO peripheral<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF. |
| Input parameter 2      | Pins: indicates the GPIO pins, refer to <a href="#">gpio_pins</a> for details.                               |
| Input parameter 3      | bit_state: to-be-written GPIO pin value.<br>This parameter can be 1 (TRUE) or 0 (FALSE)                      |
| Output parameter       | NA                                                                                                           |
| Return value           | NA                                                                                                           |
| Required preconditions | NA                                                                                                           |
| Called functions       | NA                                                                                                           |

**Example:**

```
gpio_bits_toggle(GPIOA, GPIO_PINS_0);
```

## 5.11.11 gpio\_port\_write function

The table below describes the function gpio\_port\_write.

**Table 268. 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 ports                                                                                             |
| Input parameter 1      | gpio_x: indicates the selected GPIO peripheral<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF. |
| Input parameter 2      | port_value: indicates the port value to write<br>This parameter can be 0x0000~0xFFFF.                        |
| Output parameter       | NA                                                                                                           |
| Return value           | NA                                                                                                           |
| Required preconditions | NA                                                                                                           |
| Called functions       | NA                                                                                                           |

**Example:**

```
gpio_port_write(GPIOA, 0xFFFF);
```

## 5.11.12 gpio\_pin\_wp\_config function

The table below describes the function gpio\_pin\_wp\_config.

**Table 269. 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<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF. |
| 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);
```

## 5.11.13 gpio\_pins\_huge\_driven\_config function

The table below describes the function gpio\_pins\_huge\_driven\_config.

**Table 270. gpio\_pins\_huge\_driven\_config function**

| Name                   | Description                                                                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | gpio_pins_huge_driven_config                                                                                                                                           |
| Function prototype     | void gpio_pins_huge_driven_config(gpio_type *gpio_x, uint16_t pins, confirm_state new_state);                                                                          |
| Function description   | Configure huge drive capability of GPIO pins                                                                                                                           |
| Input parameter 1      | gpio_x: indicates the selected GPIO peripheral<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.                                                           |
| Input parameter 2      | Pins: refer to <a href="#">gpio_pins</a> for details.                                                                                                                  |
| Input parameter 3      | new_state: indicates the state of maximum current sourcing/sinking capability<br>which is to be configured.<br>This parameter can be TRUE (enable) or FALSE (disable). |
| Output parameter       | NA                                                                                                                                                                     |
| Return value           | NA                                                                                                                                                                     |
| Required preconditions | NA                                                                                                                                                                     |
| Called functions       | NA                                                                                                                                                                     |

**Example:**

```
gpio_pins_huge_driven_config(GPIOA, GPIO_PINS_0, TRUE);
```

## 5.11.14 gpio\_pin\_mux\_config function

The table below describes the function gpio\_pin\_mux\_config.

**Table 271. gpio\_pin\_mux\_config function**

| Name                   | Description                                                                                                     |
|------------------------|-----------------------------------------------------------------------------------------------------------------|
| Function name          | gpio_pin_mux_config                                                                                             |
| Function prototype     | void gpio_pin_mux_config(gpio_type *gpio_x, gpio_pins_source_type gpio_pin_source, gpio_mux_sel_type gpio_mux); |
| Function description   | Configure GPIO pin multiplexed function                                                                         |
| Input parameter 1      | gpio_x: the selected GPIO peripheral<br>This parameter can be GPIOA, GPIOB, GPIOC, GPIOD or GPIOF.              |
| Input parameter 2      | gpio_pin_source: GPIO pin to be configured                                                                      |
| Input parameter 3      | gpio_mux: IOMUX index to be configured                                                                          |
| Output parameter       | NA                                                                                                              |
| Return value           | NA                                                                                                              |
| Required preconditions | NA                                                                                                              |
| Called functions       | NA                                                                                                              |

### gpio\_pin\_source

Set GPIO pins

|                     |             |
|---------------------|-------------|
| 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 |

**gpio\_mux:** Select IOMUX index

|            |
|------------|
| GPIO_MUX_0 |
| GPIO_MUX_1 |
| GPIO_MUX_2 |
| GPIO_MUX_3 |
| GPIO_MUX_4 |
| GPIO_MUX_5 |
| GPIO_MUX_6 |
| GPIO_MUX_7 |

**Example:**

```
gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE0, GPIO_MUX_0);
```

## 5.12 I2C interfaces

The I<sup>2</sup>C register structure i2c\_type is defined in the “at32f425\_i2c.h”:

```
/*
 * @brief type define i2c register all
 */
typedef struct
{
 } i2c_type;
```

The table below gives a list of the I<sup>2</sup>C registers

**Table 272. Summary of I<sup>2</sup>C register**

| Register | Description                             |
|----------|-----------------------------------------|
| ctrl1    | I <sup>2</sup> C Control register 1     |
| ctrl2    | I <sup>2</sup> C Control register 2     |
| oaddr1   | I <sup>2</sup> C Own address register 1 |
| oaddr2   | I <sup>2</sup> C Own address register 2 |
| clkctrl  | Timing register                         |
| timeout  | Timeout register                        |
| sts      | Status register                         |
| clr      | Status clear register                   |
| pec      | PEC register                            |
| rxdt     | Receive data register                   |
| txdt     | Transmit data register                  |

The table below gives a list of I<sup>2</sup>C library functions.

**Table 273. Summary of I<sup>2</sup>C library functions**

| Function name            | Description                                 |
|--------------------------|---------------------------------------------|
| i2c_reset                | I <sup>2</sup> C peripheral reset           |
| i2c_init                 | Set I <sup>2</sup> C bus speed              |
| i2c_own_address1_set     | Set I <sup>2</sup> C own address 1          |
| i2c_own_address2_set     | Set I <sup>2</sup> C own address 2          |
| i2c_own_address2_enable  | Enable I <sup>2</sup> C own address 2       |
| i2c_smbus_enable         | Enable Smbus mode                           |
| i2c_enable               | Enable I <sup>2</sup> C                     |
| i2c_clock_stretch_enable | Enable clock stretching capability          |
| i2c_ack_enable           | Enable ACK response                         |
| i2c_addr10_mode_enable   | Enable master transmit 10-bit address mode  |
| i2c_transfer_addr_set    | Set master transfer address (slave address) |
| i2c_transfer_addr_get    | Get slave address from master               |
| i2c_transfer_dir_set     | Set master data transfer direction          |
| i2c_transfer_dir_get     | Slave gets data transfer direction          |
| i2c_matched_addr_get     | Slave gets address match value              |

|                            |                                                |
|----------------------------|------------------------------------------------|
| i2c_auto_stop_enable       | Enable auto transmission stop conditions       |
| i2c_reload_enable          | Enable transmitted data reload mode            |
| i2c_cnt_set                | Set number of data to send/receive             |
| i2c_addr10_header_enable   | Enable 10-bit address header read timing       |
| i2c_general_call_enable    | Enable general call (broadcast address enable) |
| i2c_smbus_alert_set        | Set SMBus alert pin level                      |
| i2c_slave_data_ctrl_enable | Enable slave single-byte receive control       |
| i2c_pec_calculate_enable   | Enable PEC calculation                         |
| i2c_pec_transmit_enable    | Enable PEC transmit                            |
| i2c_pec_value_get          | Get current PEC value                          |
| i2c_timeout_set            | Set clock level timeout detection              |
| i2c_timeout_detct_set      | Set clock level timeout detect mode            |
| i2c_timeout_enable         | Enable clock level timeout detect              |
| i2c_ext_timeout_set        | Set accumulated clock stretching timeout       |
| i2c_ext_timeout_enable     | Enable accumulated clock stretching timeout    |
| i2c_interrupt_enable       | I2C interrupt enable                           |
| i2c_interrupt_get          | Get interrupt status                           |
| i2c_dma_enable             | DMA transfer enable                            |
| i2c_transmit_set           | Set master-initiated transfer                  |
| i2c_start_generate         | Generate start conditions                      |
| i2c_stop_generate          | Generate stop conditions                       |
| i2c_data_send              | Send data                                      |
| i2c_data_receive           | Receive data                                   |
| i2c_flag_get               | Get flag                                       |
| i2c_flag_clear             | Clear flag                                     |

**Table 274. I<sup>2</sup>C application-layer library functions**

| Function name             | Description                                             |
|---------------------------|---------------------------------------------------------|
| i2c_config                | I <sup>2</sup> C application initialization             |
| i2c_lowlevel_init         | I <sup>2</sup> C low-layer initialization               |
| i2c_wait_end              | I <sup>2</sup> C wait data transmit complete            |
| i2c_wait_flag             | I <sup>2</sup> C wait flag                              |
| i2c_master_transmit       | I <sup>2</sup> C master transmits data (polling mode)   |
| i2c_master_receive        | I <sup>2</sup> C master receives data (polling mode)    |
| i2c_slave_transmit        | I <sup>2</sup> C slave transmits data (polling mode)    |
| i2c_slave_receive         | I <sup>2</sup> C slave receives data (polling mode)     |
| i2c_master_transmit_int   | I <sup>2</sup> C master transmits data (interrupt mode) |
| i2c_master_receive_int    | I <sup>2</sup> C master receives data (interrupt mode)  |
| i2c_slave_transmit_int    | I <sup>2</sup> C slave transmits data (interrupt mode)  |
| i2c_slave_receive_int     | I <sup>2</sup> C slave receives data (interrupt mode)   |
| i2c_master_transmit_dma   | I <sup>2</sup> C master transmits data (DMA mode)       |
| i2c_master_receive_dma    | I <sup>2</sup> C master receives data (DMA mode)        |
| i2c_slave_transmit_dma    | I <sup>2</sup> C slave transmits data (DMA mode)        |
| i2c_slave_receive_dma     | I <sup>2</sup> C slave receives data (DMA mode)         |
| i2c_smbus_master_transmit | SMBus master sends data (polling mode)                  |

| Function name            | Description                                             |
|--------------------------|---------------------------------------------------------|
| i2c_smbus_master_receive | SMBus master receives data (polling mode)               |
| i2c_smbus_slave_transmit | SMBus slave sends data (polling mode)                   |
| i2c_smbus_slave_receive  | SMBus slave receives data (polling mode)                |
| i2c_memory_write         | I <sup>2</sup> C writes data to EEPROM (polling mode)   |
| i2c_memory_write_int     | I <sup>2</sup> C writes data to EEPROM (interrupt mode) |
| i2c_memory_write_dma     | I <sup>2</sup> C writes data to EEPROM (DMA mode)       |
| i2c_memory_read          | I <sup>2</sup> C reads from EEPROM (polling mode)       |
| i2c_memory_read_int      | I <sup>2</sup> C reads from EEPROM (interrupt mode)     |
| i2c_memory_read_dma      | I <sup>2</sup> C reads from EEPROM (DMA mode)           |
| i2c_evt_irq_handler      | I <sup>2</sup> C event interrupt function               |
| i2c_err_irq_handler      | I <sup>2</sup> C error interrupt function               |
| i2c_dma_tx_irq_handler   | I <sup>2</sup> C DMA Tx interrupt function              |
| i2c_dma_rx_irq_handler   | I <sup>2</sup> C DMA Rx interrupt function              |

### 5.12.1 i2c\_reset function

The table below describes the function i2c\_reset.

Table 275. i2c\_reset function

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_reset                                                                                                                  |
| Function prototype     | void i2c_reset(i2c_type *i2c_x);                                                                                           |
| Function description   | Reset all I <sup>2</sup> C registers to their initial values through CRM (Clock and reset management)                      |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| 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\_init function

The table below describes the function i2c\_init.

**Table 276. i2c\_init function**

| Name                   | Description                                                                                                                                                                                                                            |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_init                                                                                                                                                                                                                               |
| Function prototype     | void i2c_init(i2c_type *i2c_x, uint8_t dfilters, uint32_t clk);                                                                                                                                                                        |
| Function description   | Set I <sup>2</sup> C bus speed and digital filter                                                                                                                                                                                      |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2.                                                                                                             |
| Input parameter 2      | Dfilters: digital filter, ranging from 0x00 to 0x0F<br>When in use, it is recommended to program the digital filter with a maximum value to effectively filter disturbance                                                             |
| Input parameter 3      | Clk: timing register (I2C_CLKCTRL) value used to control I <sup>2</sup> C communication speed. This value can be calculated through “Artery_I2C_Timing_Configuration” defined in the <i>AN0091_AT32F435_437_I2C_Application_Note</i> . |
| Output parameter       | NA                                                                                                                                                                                                                                     |
| Return value           | NA                                                                                                                                                                                                                                     |
| Required preconditions | NA                                                                                                                                                                                                                                     |
| Called functions       | NA                                                                                                                                                                                                                                     |

**Example:**

```
i2c_init(I2C1, 0x0F, 0x80504C4E);
```

## 5.12.3 i2c\_own\_address1\_set function

The table below describes the function i2c\_own\_address1\_set.

**Table 277. 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<br>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);
```

### 5.12.4 i2c\_own\_address2\_set function

The table below describes the function i2c\_own\_address2\_set.

**Table 278. 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, i2c_addr2_mask_type mask);                                                      |
| Function description   | Set own address 2. The address 2 becomes active only after it is enabled.<br>Note: only 7-bit address is supported, not 10-bit address mode |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2.                  |
| Input parameter 2      | Address: own address 2                                                                                                                      |
| Input parameter 3      | Mask: own address 2 bit mask<br>Refer to the following "mask" descriptions for details.                                                     |
| Output parameter       | NA                                                                                                                                          |
| Return value           | NA                                                                                                                                          |
| Required preconditions | NA                                                                                                                                          |
| Called functions       | NA                                                                                                                                          |

#### mask

Own address 2 bit mask.

|                   |                                                           |
|-------------------|-----------------------------------------------------------|
| I2C_ADDR2_NOMASK: | match address bit [7:1]                                   |
| I2C_ADDR2_MASK01: | match address bit [7:2] only                              |
| I2C_ADDR2_MASK02: | match address bit [7:3] only                              |
| I2C_ADDR2_MASK03: | match address bit [7:4] only                              |
| I2C_ADDR2_MASK04: | match address bit [7:5] only                              |
| I2C_ADDR2_MASK05: | match address bit [7:6] only                              |
| I2C_ADDR2_MASK06: | match address bit [7] only                                |
| I2C_ADDR2_MASK07: | All non-I <sup>2</sup> C reserved addresses would respond |

#### Example:

```
i2c_own_address2_set(I2C1, 0xB0, I2C_ADDR2_NOMASK);
```

### 5.12.5 i2c\_own\_address2\_enable function

The table below describes the function i2c\_own\_address2\_enable.

**Table 279. 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.<br>Note that this function should be used in conjunction with the i2c_own_address2_set. |
| Input parameter 1    | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2.                                         |
| Input parameter 2    | new_state: indicates address 2 status<br>This parameter can be TRUE or FALSE.                                                                                      |

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

**Example:**

|                                      |
|--------------------------------------|
| i2c_own_address2_enable(I2C1, TRUE); |
|--------------------------------------|

### 5.12.6 i2c\_smbus\_enable function

The table below describes the function i2c\_smbus\_enable.

**Table 280. i2c\_smbus\_enable function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_smbus_enable                                                                                                           |
| Function prototype     | void i2c_smbus_enable(i2c_type *i2c_x, i2c_smbus_mode_type mode, confirm_state new_state);                                 |
| Function description   | Enable SMBus mode. After power-on reset, the default mode is I <sup>2</sup> C mode.                                        |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Mode: SMBus mode selection<br>Refer to the following “mode” descriptions for details.                                      |
| Input parameter 3      | new_state: indicates SMBus mode status<br>This parameter can be TRUE or FALSE.                                             |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**mode**

SMBus mode

I2C\_SMBUS\_MODE\_DEVICE: SMBus device

I2C\_SMBUS\_MODE\_HOST: SMBus host

**Example:**

|                                                      |
|------------------------------------------------------|
| i2c_smbus_enable(I2C1, I2C_SMBUS_MODE_DEVICE, TRUE); |
|------------------------------------------------------|

## 5.12.7 i2c\_enable function

The table below describes the function i2c\_enable.

**Table 281. 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 I <sup>2</sup> C peripheral                                                                                         |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | new_state: indicates I <sup>2</sup> C status<br>This parameter can be TRUE or FALSE.                                       |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

|                         |
|-------------------------|
| i2c_enable(I2C1, TRUE); |
|-------------------------|

## 5.12.8 i2c\_clock\_stretch\_enable function

The table below describes the function i2c\_clock\_stretch\_enable.

**Table 282. 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.<br><br>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 I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2.                                                                                                                                                                                                                                                                                                                                                                                                 |
| Input parameter 2      | new_state: indicates clock stretching status<br>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); |
|---------------------------------------|

### 5.12.9 i2c\_ack\_enable function

The table below describes the function i2c\_ack\_enable.

**Table 283. 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 I<sup>2</sup>C communication protocol, refer to I<sup>2</sup>C protocol or AT32 reference manual.</p> |
| Input parameter 1      | <p>i2c_x: indicates the selected I<sup>2</sup>C peripheral</p> <p>This parameter can be I<sup>2</sup>C1 or I<sup>2</sup>C2.</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);
```

### 5.12.10 i2c\_addr10\_mode\_enable function

The table below describes the function i2c\_addr10\_mode\_enable.

**Table 284. i2c\_addr10\_mode\_enable function**

| Name                   | Description                                                                                                                     |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_addr10_mode_enable                                                                                                          |
| Function prototype     | void i2c_addr10_mode_enable(i2c_type *i2c_x, confirm_state new_state);                                                          |
| Function description   | Enable master transmit 10-bit address mode                                                                                      |
| Input parameter 1      | <p>i2c_x: indicates the selected I<sup>2</sup>C peripheral</p> <p>This parameter can be I<sup>2</sup>C1 or I<sup>2</sup>C2.</p> |
| Input parameter 2      | <p>new_state: 10-bit address mode enable state</p> <p>This parameter can be TRUE or FALSE.</p>                                  |
| Output parameter       | NA                                                                                                                              |
| Return value           | NA                                                                                                                              |
| Required preconditions | NA                                                                                                                              |
| Called functions       | NA                                                                                                                              |

**Example:**

```
i2c_addr10_mode_enable(I2C1, TRUE);
```

### 5.12.11 i2c\_transfer\_addr\_set function

The table below describes the function i2c\_transfer\_addr\_set.

**Table 285. i2c\_transfer\_addr\_set function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_transfer_addr_set                                                                                                      |
| Function prototype     | void i2c_transfer_addr_set(i2c_type *i2c_x, uint16_t address);                                                             |
| Function description   | Set master transfer address (slave address)                                                                                |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Address: slave address                                                                                                     |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_transfer_addr_set(I2C1, 0xA0);
```

### 5.12.12 i2c\_transfer\_addr\_get function

The table below describes the function i2c\_transfer\_addr\_get.

**Table 286. i2c\_transfer\_addr\_get function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_transfer_addr_get                                                                                                      |
| Function prototype     | uint16_t i2c_transfer_addr_get(i2c_type *i2c_x);                                                                           |
| Function description   | Get slave address sent from master                                                                                         |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Output parameter       | NA                                                                                                                         |
| Return value           | uint16_t: slave address sent from master                                                                                   |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_transfer_addr_get(I2C1);
```

### 5.12.13 i2c\_transfer\_dir\_set function

The table below describes the function i2c\_transfer\_dir\_set.

**Table 287. i2c\_transfer\_dir\_set function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_transfer_dir_set                                                                                                       |
| Function prototype     | void i2c_transfer_dir_set(i2c_type *i2c_x, i2c_transfer_dir_type i2c_direction);                                           |
| Function description   | Set master data transfer direction                                                                                         |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Direction: data transfer direction<br>Refer to the following "direction" descriptions for details.                         |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

#### direction

Data transfer direction

I2C\_DIR\_TRANSMIT: Master sends data

I2C\_DIR\_RECEIVE: Master receives data

#### Example:

```
i2c_transfer_dir_set(I2C1, I2C_DIR_TRANSMIT);
```

### 5.12.14 i2c\_transfer\_dir\_get function

The table below describes the function i2c\_transfer\_dir\_get.

**Table 288. i2c\_transfer\_dir\_get function**

| Name                   | Description                                                                                                                      |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_transfer_dir_get                                                                                                             |
| Function prototype     | i2c_transfer_dir_type i2c_transfer_dir_get(i2c_type *i2c_x);                                                                     |
| Function description   | Get slave data transfer direction                                                                                                |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2.       |
| Output parameter       | NA                                                                                                                               |
| Return value           | i2c_transfer_dir_type: slave data transfer direction<br>Refer to the following "i2c_transfer_dir_type" descriptions for details. |
| Required preconditions | NA                                                                                                                               |
| Called functions       | NA                                                                                                                               |

#### i2c\_transfer\_dir\_type

Data transfer direction.

I2C\_DIR\_TRANSMIT: master sends data, and slave receives data

I2C\_DIR\_RECEIVE: master receives data and slave sends data

#### Example:

```
i2c_transfer_dir_get(I2C1);
```

### 5.12.15 i2c\_matched\_addr\_get function

The table below describes the function i2c\_matched\_addr\_get.

**Table 289. i2c\_matched\_addr\_get function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_matched_addr_get                                                                                                       |
| Function prototype     | uint8_t i2c_matched_addr_get(i2c_type *i2c_x);                                                                             |
| Function description   | Get slave address match value                                                                                              |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Output parameter       | NA                                                                                                                         |
| Return value           | uint8_t: slave matched address                                                                                             |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_matched_addr_get(I2C1);
```

### 5.12.16 i2c\_auto\_stop\_enable function

The table below describes the function i2c\_auto\_stop\_enable.

**Table 290. i2c\_auto\_stop\_enable function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_auto_stop_enable                                                                                                       |
| Function prototype     | void i2c_auto_stop_enable(i2c_type *i2c_x, confirm_state new_state);                                                       |
| Function description   | Enable auto transmit stop conditions                                                                                       |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | new_state: auto transmit stop condition enable state<br>This parameter can be TRUE or FALSE.                               |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_auto_stop_enable(I2C1, TRUE);
```

### 5.12.17 i2c\_reload\_enable function

The table below describes the function i2c\_reload\_enable.

**Table 291. i2c\_reload\_enable function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_reload_enable                                                                                                          |
| Function prototype     | void i2c_reload_enable(i2c_type *i2c_x, confirm_state new_state);                                                          |
| Function description   | Enable transmitted data reload mode                                                                                        |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | new_state: reload mode enable state<br>This parameter can be TRUE or FALSE.                                                |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_reload_enable(I2C1, TRUE);
```

### 5.12.18 i2c\_cnt\_set function

The table below describes the function i2c\_cnt\_set.

**Table 292. i2c\_cnt\_set function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_cnt_set                                                                                                                |
| Function prototype     | void i2c_cnt_set(i2c_type *i2c_x, uint8_t cnt);                                                                            |
| Function description   | Set the number of data to send or receive, ranging from 1 to 255                                                           |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Cnt: number of data to send/receive                                                                                        |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_cnt_set(I2C1, 200);
```

### 5.12.19 i2c\_addr10\_header\_enable function

The table below describes the function i2c\_addr10\_header\_enable.

**Table 293. i2c\_addr10\_header\_enable function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_addr10_header_enable                                                                                                   |
| Function prototype     | void i2c_addr10_header_enable(i2c_type *i2c_x, confirm_state new_state);                                                   |
| Function description   | Enable 10-bit address header read timing                                                                                   |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | new_state: enable state of auto transmit stop conditions<br>This parameter can be TRUE or FALSE.                           |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_addr10_header_enable(I2C1, TRUE);
```

### 5.12.20 i2c\_general\_call\_enable function

The table below describes the function i2c\_dma\_enable.

**Table 294. 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 broadcast address. After enabled, broadcast address 0x00 is responded                                               |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | new_state: Broadcast address enable state<br>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);
```

### 5.12.21 i2c\_smbus\_alert\_set function

The table below describes the function i2c\_smbus\_alert\_set.

**Table 295. 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 level (high or low)                                                                                    |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | level: SMBus alert pin level<br>Refer to the following "level" descriptions for details.                                   |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**source**

SMBus alert pin level

I2C\_SMBUS\_ALERT\_LOW: SMBus alert pin output low

I2C\_SMBUS\_ALERT\_HIGH: SMBus alert pin output high

**Example:**

```
i2c_smbus_alert_set(I2C1, I2C_SMBUS_ALERT_LOW);
```

### 5.12.22 i2c\_slave\_data\_ctrl\_enable function

The table below describes the function i2c\_start\_generate.

**Table 296. i2c\_start\_generate function**

| Name                   | Description                                                                                                                                                              |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_slave_data_ctrl_enable                                                                                                                                               |
| Function prototype     | void i2c_slave_data_ctrl_enable(i2c_type *i2c_x, confirm_state new_state);                                                                                               |
| Function description   | Enable slave data receive control. This function is used to control ACK or NACK response to each received byte when in slave receive mode. It is usually used for SMBus. |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2.                                               |
| Input parameter 2      | new_state: enable state<br>This parameter can be TRUE or FALSE.                                                                                                          |
| Output parameter       | NA                                                                                                                                                                       |
| Return value           | NA                                                                                                                                                                       |
| Required preconditions | NA                                                                                                                                                                       |
| Called functions       | NA                                                                                                                                                                       |

**Example:**

```
i2c_slave_data_ctrl_enable(I2C1, FALSE);
```

### 5.12.23 i2c\_pec\_calculate\_enable function

The table below describes the function i2c\_pec\_calculate\_enable

**Table 297. i2c\_pec\_calculate\_enable**

| 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 I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | new_state: PEC calculation state<br>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);
```

### 5.12.24 i2c\_pec\_transmit\_enable function

The table below describes the function i2c\_pec\_transmit\_enable.

**Table 298. 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   | PEC transmit enable (send/receive PEC)                                                                                     |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | new_state: PEC transmit enable state<br>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);
```

## 5.12.25 i2c\_pec\_value\_get function

The table below describes the function i2c\_pec\_value\_get

**Table 299. 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 current PEC value                                                                                                      |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Output parameter       | uint8_t: current PEC value                                                                                                 |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
pec_value = i2c_pec_value_get(I2C1);
```

## 5.12.26 i2c\_timeout\_set function

The table below describes the function i2c\_timeout\_set.

**Table 300. i2c\_timeout\_set function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_timeout_set                                                                                                            |
| Function prototype     | void i2c_timeout_set(i2c_type *i2c_x, uint16_t timeout);                                                                   |
| Function description   | Set SCL line level timeout detect time                                                                                     |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Timeout: timeout value, ranging from 0x0000 to 0x0FFF                                                                      |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_timeout_set(I2C1, 0xFFFF);
```

### 5.12.27 i2c\_timeout\_detcet\_set function

The table below describes the function i2c\_timeout\_detcet\_set.

**Table 301. i2c\_timeout\_detcet\_set function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_timeout_detcet_set                                                                                                     |
| Function prototype     | void i2c_timeout_detcet_set(i2c_type *i2c_x, i2c_timeout_detcet_type mode);                                                |
| Function description   | Set SCL line level timeout detect mode                                                                                     |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Mode: level detect mode<br>Refer to the following "mode" descriptions for details.                                         |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

#### mode

Level detection mode.

I2C\_TIMEOUT\_DETCET\_HIGH: High level timeout detect

I2C\_TIMEOUT\_DETCET\_LOW: Low level timeout detect

#### Example:

```
i2c_timeout_detcet_set(I2C1, I2C_TIMEOUT_DETCET_HIGH);
```

### 5.12.28 i2c\_timeout\_enable function

The table below describes the function i2c\_timeout\_enable.

**Table 302. i2c\_timeout\_enable function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_timeout_enable                                                                                                         |
| Function prototype     | void i2c_timeout_enable(i2c_type *i2c_x, confirm_state new_state);                                                         |
| Function description   | Enable SCL line level timeout detect                                                                                       |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | new_state: level timeout detect enable state<br>This parameter can be TRUE or FALSE.                                       |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

#### Example:

```
i2c_timeout_enable(I2C1, TRUE);
```

## 5.12.29 i2c\_ext\_timeout\_set function

The table below describes the function i2c\_ext\_timeout\_set.

**Table 303. i2c\_ext\_timeout\_set function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_ext_timeout_set                                                                                                        |
| Function prototype     | void i2c_ext_timeout_set(i2c_type *i2c_x, uint16_t timeout);                                                               |
| Function description   | Set SCL line cumulative clock stretching timeout value, usually used in SMBus mode                                         |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Timeout: range from 0x0000 to 0x0FFF                                                                                       |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_ext_timeout_set(I2C1, 0x0FFF);
```

## 5.12.30 i2c\_ext\_timeout\_enable function

The table below describes the function i2c\_ext\_timeout\_enable.

**Table 304. i2c\_ext\_timeout\_enable function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_ext_timeout_enable                                                                                                     |
| Function prototype     | void i2c_ext_timeout_enable(i2c_type *i2c_x, confirm_state new_state);                                                     |
| Function description   | Enable SCL line cumulative clock stretching timeout                                                                        |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | new_state: cumulative clock stretching timeout enable state<br>This parameter can be TRUE or FALSE.                        |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_ext_timeout_enable(I2C1, TRUE);
```

### 5.12.31 i2c\_interrupt\_enable function

The table below describes the function i2c\_interrupt\_enable.

Table 305. i2c\_interrupt\_enable function

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_interrupt_enable                                                                                                       |
| Function prototype     | void i2c_interrupt_enable(i2c_type *i2c_x, uint32_t source, confirm_state new_state);                                      |
| Function description   | I2C interrupt enable                                                                                                       |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Source: interrupt sources<br>Refer to the following "source" descriptions for details.                                     |
| Input parameter 3      | new_state: interrupt enable state<br>This parameter can be TRUE or FALSE.                                                  |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

#### source

Interrupt source.

|                  |                                              |
|------------------|----------------------------------------------|
| I2C_TD_INT:      | Data transmit interrupt                      |
| I2C_RD_INT:      | Data receive interrupt                       |
| I2C_ADDR_INT:    | Address match interrupt                      |
| I2C_ACKFIAL_INT: | Acknowledge failure interrupt                |
| I2C_STOP_INT:    | Stop condition generation complete interrupt |
| I2C_TDC_INT:     | Data transfer complete interrupt             |
| I2C_ERR_INT:     | Error interrupt                              |

#### Example:

```
i2c_interrupt_enable(I2C1, I2C_TD_INT, TRUE);
```

### 5.12.32 i2c\_interrupt\_get function

The table below describes the function i2c\_interrupt\_get.

**Table 306. i2c\_interrupt\_get function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_interrupt_get                                                                                                          |
| Function prototype     | flag_status i2c_interrupt_get(i2c_type *i2c_x, uint16_t source);                                                           |
| Function description   | Get interrupt enable state                                                                                                 |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Source: interrupt source<br>Refer to the following "source" descriptions for details.                                      |
| Output parameter       | flag_status: flag status<br>This parameter can be SET or RESET.                                                            |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

#### source

Interrupt source.

|                  |                                              |
|------------------|----------------------------------------------|
| I2C_TD_INT:      | Data transmit interrupt                      |
| I2C_RD_INT:      | Data receive interrupt                       |
| I2C_ADDR_INT:    | Address match interrupt                      |
| I2C_ACKFAIL_INT: | Acknowledge failure interrupt                |
| I2C_STOP_INT:    | Stop condition generation complete interrupt |
| I2C_TDC_INT:     | Data transfer complete interrupt             |
| I2C_ERR_INT:     | Error interrupt                              |

#### Example:

```
i2c_interrupt_get(I2C1, I2C_TD_INT, TRUE);
```

### 5.12.33 i2c\_dma\_enable function

The table below describes the function i2c\_dma\_enable.

**Table 307. i2c\_dma\_enable function**

| Name                 | Description                                                                                                                |
|----------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name        | i2c_dma_enable                                                                                                             |
| Function prototype   | void i2c_dma_enable(i2c_type *i2c_x, i2c_dma_request_type dma_req, confirm_state new_state);                               |
| Function description | DMA transfer enable                                                                                                        |
| Input parameter 1    | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2    | dma_req: DMA request<br>Refer to the following "dma_req" descriptions for details.                                         |
| Input parameter 3    | new_state: DMA enable state<br>This parameter can be TRUE or FALSE.                                                        |
| Output parameter     | NA                                                                                                                         |
| Return value         | NA                                                                                                                         |

| Name                   | Description |
|------------------------|-------------|
| Required preconditions | NA          |
| Called functions       | NA          |

**dma\_req**

I2C\_DMA\_REQUEST\_TX: DMA data transmit enable

I2C\_DMA\_REQUEST\_RX: DMA data receive enable

**Example:**

|                                                 |
|-------------------------------------------------|
| i2c_dma_enable(I2C1, I2C_DMA_REQUEST_TX, TRUE); |
|-------------------------------------------------|

### 5.12.34 i2c\_transmit\_set function

The table below describes the function i2c\_transmit\_set.

**Table 308. i2c\_transmit\_set function**

| Name                   | Description                                                                                                                           |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_transmit_set                                                                                                                      |
| Function prototype     | void i2c_transmit_set(i2c_type *i2c_x, uint16_t address, uint8_t cnt, i2c_reload_stop_mode_type rld_stop, i2c_start_mode_type start); |
| Function description   | Set master transmit. This function is used to start data transfer on bus.                                                             |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2.            |
| Input parameter 2      | Address: slave address                                                                                                                |
| Input parameter 3      | Cnt: count of data to send/receive                                                                                                    |
| Input parameter 4      | rld_stop: reload mode and STOP condition generation mode<br>Refer to the following "rld_stop" descriptions for details.               |
| Input parameter 5      | Start: set START condition generation mode<br>Refer to the following "start" descriptions for details.                                |
| Output parameter       | NA                                                                                                                                    |
| Return value           | NA                                                                                                                                    |
| Required preconditions | NA                                                                                                                                    |
| Called functions       | NA                                                                                                                                    |

**rld\_stop**

Reload mode and STOP condition generation mode.

I2C\_AUTO\_STOP\_MODE: Auto stop mode (automatically sends STOP condition)

I2C\_SOFT\_STOP\_MODE: Software stop mode (software sends STOP condition, usually RESTART condition)

I2C\_RELOAD\_MODE: Reload mode (when a single transfer &gt;255)

**start**

START condition generation mode.

I2C\_WITHOUT\_START: Start sending data, without START, used in reload mode

I2C\_GEN\_START\_READ: Start sending data, with START condition (for master receive data)

I2C\_GEN\_START\_WRITE: Start sending data with START condition (for master transmit data)

**Example:**

|                                                                  |
|------------------------------------------------------------------|
| i2c_transmit_set(I2C1, I2C_AUTO_STOP_MODE, I2C_GEN_START_WRITE); |
|------------------------------------------------------------------|

### 5.12.35 i2c\_start\_generate function

The table below describes the function i2c\_start\_generate.

**Table 309. i2c\_slave\_transmit function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_start_generate                                                                                                         |
| Function prototype     | void i2c_start_generate(i2c_type *i2c_x);                                                                                  |
| Function description   | Generate a START condition (for master)                                                                                    |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_start_generate(I2C1);
```

### 5.12.36 i2c\_stop\_generate function

The table below describes the function i2c\_stop\_generate.

**Table 310. i2c\_stop\_generate function**

| Name                   | Description                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_stop_generate                                                                                                          |
| Function prototype     | void i2c_stop_generate(i2c_type *i2c_x);                                                                                   |
| Function description   | Generate a STOP condition                                                                                                  |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_stop_generate(I2C1);
```

### 5.12.37 i2c\_data\_send function

The table below describes the function i2c\_data\_send.

Table 311. 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 I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Data: data to be sent                                                                                                      |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
i2c_data_send(I2C1, 0x55);
```

### 5.12.38 i2c\_data\_receive function

The table below describes the function i2c\_data\_receive

Table 312. 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 I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Output parameter       | uint8_t: data to be received                                                                                               |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
data_value = i2c_data_receive(I2C1);
```

### 5.12.39 i2c\_flag\_get function

The table below describes the function i2c\_flag\_get

**Table 313. 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 I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Flag: the selected flag<br>Refer to the following "flag" descriptions for details.                                         |
| Output parameter       | NA                                                                                                                         |
| Return value           | flag_status: flag status<br>This parameter can be SET or RESET.                                                            |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

#### flag

This bit is used to select a flag to get its status. Optional parameters are below:

|                   |                                            |
|-------------------|--------------------------------------------|
| I2C_TDBE_FLAG:    | Transmit data register empty flag          |
| I2C_TDIS_FLAG:    | Transmit interrupt status flag             |
| I2C_RDBF_FLAG:    | Receive data buffer full flag              |
| I2C_ADDRF_FLAG:   | Address match flag                         |
| I2C_ACKFAIL_FLAG: | Acknowledge failure flag                   |
| I2C_STOPF_FLAG:   | STOP condition generation complete flag    |
| I2C_TDC_FLAG:     | Data transfer complete flag                |
| I2C_TCRLD_FLAG:   | Transfer complete to wait for loading data |
| I2C_BUSERR_FLAG:  | Bus error flag                             |
| I2C_ARLOST_FLAG:  | Arbitration lost flag                      |
| I2C_OUF_FLAG:     | Overflow or underflow flag                 |
| I2C_PECERR_FLAG:  | PEC receive error flag                     |
| I2C_TMOUT_FLAG:   | SMBus timeout flag                         |
| I2C_ALERTF_FLAG:  | SMBus alert flag                           |
| I2C_BUSYF_FLAG:   | Bus busy flag                              |
| I2C_SDIR_FLAG:    | Slave data transfer direction              |

#### Example:

```
i2c_flag_get(I2C1, I2C_TDIS_FLAG);
```

## 5.12.40 i2c\_interrupt\_flag\_get function

The table below describes the function i2c\_interrupt\_flag\_get.

**Table 314. 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 I2C interrupt flag status, and check corresponding interrupt enable bit                                                |
| Input parameter 1      | i2c_x: indicates the selected I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Flag: the selected flag<br>Refer to the “flag” below for details.                                                          |
| Output parameter       | NA                                                                                                                         |
| Return value           | flag_status: SET or RESET                                                                                                  |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

### flag

This bit is used to select a flag, including:

- I2C\_TDBE\_FLAG: Transmit data register empty flag
- I2C\_TDIS\_FLAG: Transmit interrupt status flag
- I2C\_RDBF\_FLAG: Receive data buffer full flag
- I2C\_ADDRF\_FLAG: Address match flag
- I2C\_ACKFAIL\_FLAG: Acknowledge failure flag
- I2C\_STOPF\_FLAG: STOP condition generation complete flag
- I2C\_TDC\_FLAG: Data transfer complete flag
- I2C\_TCRLD\_FLAG: Transfer complete to wait for loading data
- I2C\_BUSERR\_FLAG: Bus error flag
- I2C\_ARLOST\_FLAG: Arbitration lost flag
- I2C\_OUF\_FLAG: Overflow or underflow 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_TDIS_FLAG);
```

### 5.12.41 i2c\_flag\_clear function

The table below describes the function i2c\_flag\_clear.

Table 315. 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 I <sup>2</sup> C peripheral<br>This parameter can be I <sup>2</sup> C1 or I <sup>2</sup> C2. |
| Input parameter 2      | Flag: the selected flag<br>Refer to the following "flag" descriptions for details.                                         |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

#### flag

This bit is used to select a flag, including:

- I2C\_ADDRF\_FLAG: Address match flag
- I2C\_ACKFAIL\_FLAG: Acknowledge failure flag
- I2C\_STOPF\_FLAG: STOP condition generation complete flag
- I2C\_BUSERR\_FLAG: Bus error flag
- I2C\_ARLOST\_FLAG: Arbitration lost flag
- I2C\_OUF\_FLAG: Overflow or underflow flag
- I2C\_PECERR\_FLAG: PEC receive error flag
- I2C\_TMOUT\_FLAG: SMBus timeout flag
- I2C\_ALERTF\_FLAG: SMBus alert flag

#### Example:

```
i2c_flag_clear(I2C1, I2C_ACKFAIL_FLAG);
```

## 5.12.42 i2c\_config function

The table below describes the function i2c\_config.

**Table 316. i2c\_config function**

| Name                   | Description                                                                                                                                                                                       |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_config                                                                                                                                                                                        |
| Function prototype     | void i2c_config(i2c_handle_type* hi2c);                                                                                                                                                           |
| Function description   | I <sup>2</sup> C initialization function used to initialize I <sup>2</sup> C. Call the function i2c_lowlevel_init() to initialize I <sup>2</sup> C peripherals, GPIO, DMA, interrupts and others. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                       |
| Output parameter       | NA                                                                                                                                                                                                |
| Return value           | NA                                                                                                                                                                                                |
| Required preconditions | NA                                                                                                                                                                                                |
| Called functions       | NA                                                                                                                                                                                                |

### i2c\_handle\_type\* hi2c

i2c\_handle\_type is defined in the i2c\_application.h.

typedef struct

```
{
 i2c_type *i2cx;
 uint8_t *pbuff;
 __IO uint16_t psize;
 __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

Select an I<sup>2</sup>C peripheral from I<sup>2</sup>C1, I<sup>2</sup>C2 or I<sup>2</sup>C3

### pbuff

An array of data to be sent or received.

### psize

This bit is used to count the size of bytes in a single transfer when the transfer size is over 255. It is used in internal state machine. Users don't care.

### pcount

The number of data to be sent or received.

### mode

I<sup>2</sup>C communication mode. It is used in internal state machine. Users don't care.

### timeout

Communications timeout

### status

Transfer status. It is used in internal state machine. Users don't care.

#### **error\_code**

This bit is used to enumerate error code in the i2c\_status\_type. When a communication error occurred, it logs the corresponding error code.

|                    |                                                 |
|--------------------|-------------------------------------------------|
| 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_TCRLD:     | Wait for TCRLD timeout                          |
| I2C_ERR_TDC:       | Wait for TDC timeout                            |
| I2C_ERR_ADDR:      | Address send error                              |
| I2C_ERR_STOP:      | STOP condition send error                       |
| I2C_ERR_ACKFAIL:   | Acknowledge error                               |
| I2C_ERR_TIMEOUT:   | Timeout error                                   |
| I2C_ERR_INTERRUPT: | Enter an interrupt when an error event occurred |

#### **dma\_tx\_channel**

I2C transmit DMA channel

#### **dma\_rx\_channel**

I2C receive DMA channel

#### **dma\_init\_struct**

DMA initialization structure

#### **Example:**

```
i2c_handle_type hi2c;
hi2c.i2cx = I2C1;
i2c_config(&hi2c);
```

### 5.12.43 i2c\_lowlevel\_init function

The table below describes the function i2c\_lowlevel\_init.

**Table 317. i2c\_lowlevel\_init function**

| Name                   | Description                                                                                                                                                                                                                                               |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_lowlevel_init                                                                                                                                                                                                                                         |
| Function prototype     | void i2c_lowlevel_init(i2c_handle_type* hi2c);                                                                                                                                                                                                            |
| Function description   | I <sup>2</sup> C lower-level initialization callback function. It is called in the i2c_config to initialize I <sup>2</sup> C peripherals, GPIO, DMA, interrupts, etc. It requires users to implement I <sup>2</sup> C initialization inside the function. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                               |
| 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)
 {
 Implement I2C1 initialization
 }
 else if(hi2c->i2cx == I2C2)
 {
 Implement I2C1 initialization
 }
}
```

### 5.12.44 i2c\_wait\_end function

The table below describes the function i2c\_wait\_end.

**Table 318. 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 end of communications. This function is used in DMA and interrupt transfer modes as they are non-blocking functions and can thus be used to wait for the end of transfer. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                            |
| Input parameter 2      | Timeout: wait timeout                                                                                                                                                                  |
| Output parameter       | NA                                                                                                                                                                                     |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                           |
| Required preconditions | NA                                                                                                                                                                                     |

| Name             | Description |
|------------------|-------------|
| 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 transfer*/
if(i2c_wait_end(&hi2c, 0xFFFFFFFF) != I2C_OK)
{
 error_handler(i2c_status);
}

```

## 5.12.45 i2c\_wait\_flag function

The table below describes the function i2c\_wait\_flag.

**Table 319. 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 a flag to be set or reset<br>Only BUSFY flag is “wait for a flag to be reset”, and others are “wait for a flag to be set”         |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                |
| Input parameter 2      | Flag: the selected flag<br>Refer to the following “flag” descriptions for details.                                                         |
| Input parameter 3      | event_check: check if the event has occurred or not while waiting for a flag<br>Refer to the “event_check” descriptions below for details. |
| Input parameter 4      | Timeout: wait timeout                                                                                                                      |
| Output parameter       | NA                                                                                                                                         |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                               |
| Required preconditions | NA                                                                                                                                         |
| Called functions       | NA                                                                                                                                         |

**flag**

Select a flag to wait for.

- |                   |                                         |
|-------------------|-----------------------------------------|
| I2C_TDBE_FLAG:    | Transmit data register empty flag       |
| I2C_TDIS_FLAG:    | Transmit interrupt status flag          |
| I2C_RDBF_FLAG:    | Receive data buffer full flag           |
| I2C_ADDRF_FLAG:   | Address match flag                      |
| I2C_ACKFAIL_FLAG: | Acknowledge failure flag                |
| I2C_STOPF_FLAG:   | STOP condition generation complete flag |
| I2C_TDC_FLAG:     | Data transfer complete flag             |

|                  |                                            |
|------------------|--------------------------------------------|
| I2C_TCRLD_FLAG:  | Transfer complete to wait for loading data |
| I2C_BUSERR_FLAG: | Bus error flag                             |
| I2C_ARLOST_FLAG: | Arbitration lost flag                      |
| I2C_OUF_FLAG:    | Overflow or underflow flag                 |
| I2C_PECERR_FLAG: | PEC receive error flag                     |
| I2C_TMOUT_FLAG:  | SMBus timeout flag                         |
| I2C_ALERTF_FLAG: | SMBus alert flag                           |
| I2C_BUSYF_FLAG:  | Bus busy flag                              |
| I2C_SDIR_FLAG:   | Slave data transfer direction              |

**event\_check**

Check if the event has occurred or not while waiting for a flag.

I2C\_EVENT\_CHECK\_NONE: None

I2C\_EVENT\_CHECK\_ACKFAIL: Check ACKFAIL event

I2C\_EVENT\_CHECK\_STOP: Check STOP event

**Example:**

```
i2c_wait_flag(&hi2c, I2C_BUSYF_FLAG, I2C_EVENT_CHECK_NONE, 0xFFFFFFFF);
```

### 5.12.46 i2c\_master\_transmit function

The table below describes the function i2c\_master\_transmit.

**Table 320. 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 sends data (polling mode). This is a blocking function, and so I <sup>2</sup> C transfer ends after the function is executed. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                          |
| Input parameter 2      | Address: slave address                                                                                                               |
| Input parameter 3      | Pdata: array address of to-be-sent data                                                                                              |
| Input parameter 4      | Size: the size of data to be sent                                                                                                    |
| Input parameter 5      | Timeout: wait timeout                                                                                                                |
| Output parameter       | NA                                                                                                                                   |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                         |
| Required preconditions | NA                                                                                                                                   |
| Called functions       | NA                                                                                                                                   |

**Example:**

```
i2c_master_transmit(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.47 i2c\_master\_receive function

The table below describes the function i2c\_master\_receive.

**Table 321. i2c\_master\_receivefunction**

| 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 a blocking type. After the execution is done, so does I <sup>2</sup> C transfer. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                            |
| Input parameter 2      | Address: slave address                                                                                                                 |
| Input parameter 3      | Pdata: array address to receive data                                                                                                   |
| 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<br>Refer to 5.12.42 for details.                                                                           |
| Required preconditions | NA                                                                                                                                     |
| Called functions       | NA                                                                                                                                     |

**Example:**

```
i2c_master_receive(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.48 i2c\_slave\_transmit function

The table below describes the function i2c\_slave\_transmit.

**Table 322. 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 a blocking type. In other words, after the function execution is done, so is I <sup>2</sup> C transfer. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                               |
| Input parameter 2      | Pdata: array address of data to be sent                                                                                                                   |
| Input parameter 3      | Size: number of data to be sent                                                                                                                           |
| Input parameter 4      | Timeout: wait timeout                                                                                                                                     |
| Output parameter       | NA                                                                                                                                                        |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                              |
| Required preconditions | NA                                                                                                                                                        |
| Called functions       | NA                                                                                                                                                        |

**Example:**

```
i2c_slave_transmit(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.49 i2c\_slave\_receive function

The table below describes the function i2c\_slave\_receive.

**Table 323. 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 a blocking type. In other words, after the function execution is done, so is I <sup>2</sup> C transfer. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                  |
| Input parameter 2      | Pdata: array address to receive data                                                                                                                         |
| Input parameter 3      | Size: number of data to be received                                                                                                                          |
| Input parameter 4      | Timeout: wait timeout                                                                                                                                        |
| Output parameter       | NA                                                                                                                                                           |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                 |
| Required preconditions | NA                                                                                                                                                           |
| Called functions       | NA                                                                                                                                                           |

**Example:**

```
i2c_slave_receive(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

## 5.12.50 i2c\_master\_transmit\_int function

The table below describes the function i2c\_master\_transmit\_int.

**Table 324. 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 a non-blocking type. In other words, after the function execution is done, I <sup>2</sup> C transfer has not completed yet. In this case, it is possible to call the i2c_wait_end() to wait for the completion of communication. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                                           |
| Input parameter 2      | Address: slave address                                                                                                                                                                                                                                                                |
| Input parameter 3      | Pdata: array address of data to be sent                                                                                                                                                                                                                                               |
| Input parameter 4      | Size: number of data to be sent                                                                                                                                                                                                                                                       |
| Input parameter 5      | Timeout: wait timeout                                                                                                                                                                                                                                                                 |
| Output parameter       | NA                                                                                                                                                                                                                                                                                    |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                                          |
| Required preconditions | NA                                                                                                                                                                                                                                                                                    |
| Called functions       | NA                                                                                                                                                                                                                                                                                    |

**Example:**

```
i2c_master_transmit_int(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.51 i2c\_master\_receive\_int function

The table below describes the function i2c\_master\_receive\_int.

**Table 325. 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 (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                                           |
| Input parameter 2      | Address: slave address                                                                                                                                                                                                                                                                |
| Input parameter 3      | Pdata: array address to receive data                                                                                                                                                                                                                                                  |
| Input parameter 4      | Size: number of data to be received                                                                                                                                                                                                                                                   |
| Input parameter 5      | Timeout: wait timeout                                                                                                                                                                                                                                                                 |
| Output parameter       | NA                                                                                                                                                                                                                                                                                    |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                                          |
| Required preconditions | NA                                                                                                                                                                                                                                                                                    |
| Called functions       | NA                                                                                                                                                                                                                                                                                    |

**Example:**

```
i2c_master_receive_int(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.52 i2c\_slave\_transmit\_int function

The table below describes the function i2c\_master\_receive\_int.

**Table 326. i2c\_master\_receive\_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 (through interrupt mode). This function operates in non-blocking mode. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer. |
| Input parameter 1    | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                                              |
| Input parameter 2    | Pdata: array address of data to be sent                                                                                                                                                                                                                                                  |
| Input parameter 3    | Size: number of data to be sent                                                                                                                                                                                                                                                          |
| Input parameter 4    | Timeout: wait timeout                                                                                                                                                                                                                                                                    |
| Output parameter     | NA                                                                                                                                                                                                                                                                                       |
| Return value         | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                                             |

| Name                   | Description |
|------------------------|-------------|
| Required preconditions | NA          |
| Called functions       | NA          |

**Example:**

```
i2c_slave_transmit_int(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.53 i2c\_slave\_receive\_int function

The table below describes the function i2c\_slave\_receive\_int

**Table 327. i2c\_master\_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 (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                                          |
| Input parameter 2      | Pdata: array address to receive data                                                                                                                                                                                                                                                 |
| Input parameter 3      | Size: number of data to be received                                                                                                                                                                                                                                                  |
| Input parameter 4      | Timeout: wait timeout                                                                                                                                                                                                                                                                |
| Output parameter       | NA                                                                                                                                                                                                                                                                                   |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                                         |
| Required preconditions | NA                                                                                                                                                                                                                                                                                   |
| Called functions       | NA                                                                                                                                                                                                                                                                                   |

**Example:**

```
i2c_slave_receive_int(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.54 i2c\_master\_transmit\_dma function

The table below describes the function i2c\_master\_transmit\_dma.

Table 328. 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 (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                                  |
| Input parameter 2      | Address: slave address                                                                                                                                                                                                                                                       |
| Input parameter 3      | Pdata: array address of data to be sent                                                                                                                                                                                                                                      |
| 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<br>Refer to 5.12.42 for details.                                                                                                                                                                                                                 |
| Required preconditions | NA                                                                                                                                                                                                                                                                           |
| Called functions       | NA                                                                                                                                                                                                                                                                           |

**Example:**

```
i2c_master_transmit_dma(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.55 i2c\_master\_receive\_dma function

The table below describes the function i2c\_master\_receive\_dma.

**Table 329. 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 (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                                     |
| Input parameter 2      | Address: slave address                                                                                                                                                                                                                                                          |
| Input parameter 3      | Pdata: array address to receive data                                                                                                                                                                                                                                            |
| Input parameter 4      | Size: number of data to be received                                                                                                                                                                                                                                             |
| Input parameter 5      | Timeout: wait timeout                                                                                                                                                                                                                                                           |
| Output parameter       | NA                                                                                                                                                                                                                                                                              |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                                    |
| Required preconditions | NA                                                                                                                                                                                                                                                                              |
| Called functions       | NA                                                                                                                                                                                                                                                                              |

**Example:**

```
i2c_master_receive_dma(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.56 i2c\_slave\_transmit\_dma function

The table below describes the function i2c\_slave\_transmit\_dma.

**Table 330. 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 (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer. |
| Input parameter 1    | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                                 |
| Input parameter 2    | Pdata: array address of data to be sent                                                                                                                                                                                                                                     |
| Input parameter 3    | Size: number of data to be sent                                                                                                                                                                                                                                             |
| Input parameter 4    | Timeout: wait timeout                                                                                                                                                                                                                                                       |
| Output parameter     | NA                                                                                                                                                                                                                                                                          |
| Return value         | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                                |

| Name                   | Description |
|------------------------|-------------|
| Required preconditions | NA          |
| Called functions       | NA          |

**Example:**

```
i2c_slave_transmit_dma(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.57 i2c\_slave\_receive\_dma function

The table below describes the function i2c\_slave\_transmit\_dma.

**Table 331. i2c\_slave\_transmit\_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 (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                                    |
| Input parameter 2      | Pdata: array address to receive data                                                                                                                                                                                                                                           |
| Input parameter 3      | Size: number of data to be received                                                                                                                                                                                                                                            |
| Input parameter 4      | Timeout: wait timeout                                                                                                                                                                                                                                                          |
| Output parameter       | NA                                                                                                                                                                                                                                                                             |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                                   |
| Required preconditions | NA                                                                                                                                                                                                                                                                             |
| Called functions       | NA                                                                                                                                                                                                                                                                             |

**Example:**

```
i2c_slave_receive_dma(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.58 i2c\_smbus\_master\_transmit function

The table below describes the function i2c\_smbus\_master\_transmit

**Table 332. i2c\_smbus\_master\_transmit function**

| Name                 | Description                                                                                                                                                                                                        |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name        | i2c_smbus_master_transmit                                                                                                                                                                                          |
| Function prototype   | i2c_status_type i2c_smbus_master_transmit(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);                                                                               |
| Function description | SMBus master sends data (through polling mode). This function is a blocking type. In other words, after the function execution is done, so is data transfer. It is mainly used for PEC transmission and reception. |
| Input parameter 1    | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                        |
| Input parameter 2    | Address: slave address                                                                                                                                                                                             |
| Input parameter 3    | Pdata: array address of data to be sent                                                                                                                                                                            |
| Input parameter 4    | Size: number of data to be sent                                                                                                                                                                                    |

| Name                   | Description                                                  |
|------------------------|--------------------------------------------------------------|
| Input parameter 5      | Timeout: wait timeout                                        |
| Output parameter       | NA                                                           |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details. |
| Required preconditions | NA                                                           |
| Called functions       | NA                                                           |

**Example:**

```
i2c_smbus_master_transmit(&hi2c, 0xB0, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.59 i2c\_smbus\_master\_receive function

The table below describes the function i2c\_smbus\_master\_receive

**Table 333. i2c\_smbus\_master\_receive function**

| Name                   | Description                                                                                                                                                                                                           |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_smbus_master_receive                                                                                                                                                                                              |
| Function prototype     | i2c_status_type i2c_smbus_master_receive(i2c_handle_type* hi2c, uint16_t address, uint8_t* pdata, uint16_t size, uint32_t timeout);                                                                                   |
| Function description   | SMBus master receives data (through polling mode). This function is a blocking type. In other words, after the function execution is done, so is data transfer. It is mainly used for PEC transmission and reception. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                           |
| Input parameter 2      | Address: slave address                                                                                                                                                                                                |
| Input parameter 3      | Pdata: array address to receive data                                                                                                                                                                                  |
| Input parameter 4      | Size: number of data to be received                                                                                                                                                                                   |
| Input parameter 5      | Timeout: wait timeout                                                                                                                                                                                                 |
| Output parameter       | NA                                                                                                                                                                                                                    |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                          |
| Required preconditions | NA                                                                                                                                                                                                                    |
| Called functions       | NA                                                                                                                                                                                                                    |

**Example:**

```
i2c_smbus_master_receive(&hi2c, 0xB0, rx_buf, 8, 0xFFFFFFFF);
```

## 5.12.60 i2c\_smbus\_slave\_transmit function

The table below describes the function i2c\_smbus\_slave\_transmit

**Table 334. i2c\_smbus\_slave\_transmit function**

| Name                   | Description                                                                                                                                                                                                       |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_smbus_slave_transmit                                                                                                                                                                                          |
| Function prototype     | i2c_status_type i2c_smbus_slave_transmit(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);                                                                                                 |
| Function description   | SMBus slave sends data (through polling mode). This function is a blocking type. In other words, after the function execution is done, so is data transfer. It is mainly used for PEC transmission and reception. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                       |
| Input parameter 2      | Pdata: array address of data to be sent                                                                                                                                                                           |
| Input parameter 3      | Size: number of data to be sent                                                                                                                                                                                   |
| Input parameter 4      | Timeout: wait timeout                                                                                                                                                                                             |
| Output parameter       | NA                                                                                                                                                                                                                |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                      |
| Required preconditions | NA                                                                                                                                                                                                                |
| Called functions       | NA                                                                                                                                                                                                                |

**Example:**

```
i2c_smbus_slave_transmit(&hi2c, tx_buf, 8, 0xFFFFFFFF);
```

### 5.12.61 i2c\_smbus\_slave\_receive function

The table below describes the function i2c\_smbus\_slave\_receive

**Table 335. i2c\_smbus\_slave\_receive function**

| Name                   | Description                                                                                                                                                                                                          |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_smbus_slave_receive                                                                                                                                                                                              |
| Function prototype     | i2c_status_type i2c_smbus_slave_receive(i2c_handle_type* hi2c, uint8_t* pdata, uint16_t size, uint32_t timeout);                                                                                                     |
| Function description   | SMBus slave receives data (through polling mode). This function is a blocking type. In other words, after the function execution is done, so is data transfer. It is mainly used for PEC transmission and reception. |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                          |
| Input parameter 2      | Pdata: array address to receive data                                                                                                                                                                                 |
| Input parameter 3      | Size: number of data to be received                                                                                                                                                                                  |
| Input parameter 4      | Timeout: wait timeout                                                                                                                                                                                                |
| Output parameter       | NA                                                                                                                                                                                                                   |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                         |
| Required preconditions | NA                                                                                                                                                                                                                   |
| Called functions       | NA                                                                                                                                                                                                                   |

**Example:**

```
i2c_smbus_slave_receive(&hi2c, rx_buf, 8, 0xFFFFFFFF);
```

### 5.12.62 i2c\_memory\_write function

The table below describes the function i2c\_memory\_write

**Table 336. 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 data to EEPROM (through polling mode). This function is a blocking type. In other words, after the function execution is done, so is I <sup>2</sup> C transfer.                           |
| Input parameter 1    | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                     |
| Input parameter 2    | mem_address_width: EEPROM memory address width<br>Refer to the “mem_address_width” 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 be sent                                                                                                                                                         |
| Input parameter 6    | Size: number of data to be sent                                                                                                                                                                 |
| Input parameter 7    | Timeout: wait timeout                                                                                                                                                                           |
| Output parameter     | NA                                                                                                                                                                                              |
| Return value         | i2c_status_type: error code                                                                                                                                                                     |

| Name                   | Description                   |
|------------------------|-------------------------------|
|                        | Refer to 5.12.42 for details. |
| Required preconditions | NA                            |
| Called functions       | NA                            |

**mem\_address\_width**

EEPROM memory address width

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit address width

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit address width

**Example:**

```
i2c_memory_write(&hi2c, 0xA0, 0x05, tx_buf, 8, 0xFFFFFFFF);
```

## 5.12.63 i2c\_memory\_write\_int function

The table below describes the function i2c\_memory\_write\_int

**Table 337. 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,<br>i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t<br>mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);                                                                    |
| Function description   | Write EEPROM (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                                  |
| Input parameter 2      | mem_address_width: EEPROM memory address width<br>Refer to the "mem_address_width" 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 be sent                                                                                                                                                                                                                                      |
| Input parameter 6      | size: number of data to be sent                                                                                                                                                                                                                                              |
| Input parameter 7      | Timeout: wait timeout                                                                                                                                                                                                                                                        |
| Output parameter       | NA                                                                                                                                                                                                                                                                           |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                                 |
| Required preconditions | NA                                                                                                                                                                                                                                                                           |
| Called functions       | NA                                                                                                                                                                                                                                                                           |

**mem\_address\_width**

EEPROM memory address width

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit address width

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit address width

**Example:**

```
i2c_memory_write_int(&hi2c, 0xA0, 0x05, tx_buf, 8, 0xFFFFFFFF);
```

## 5.12.64 i2c\_memory\_write\_dma function

The table below describes the function i2c\_memory\_write\_dma

**Table 338. 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,<br>i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t<br>mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);                                                              |
| Function description   | Write EEPROM (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                            |
| Input parameter 2      | mem_address_width: EEPROM memory address width<br>Refer to the “mem_address_width” 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 be sent                                                                                                                                                                                                                                |
| Input parameter 6      | size: number of data to be sent                                                                                                                                                                                                                                        |
| Input parameter 7      | Timeout: wait timeout                                                                                                                                                                                                                                                  |
| Output parameter       | NA                                                                                                                                                                                                                                                                     |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                           |
| Required preconditions | NA                                                                                                                                                                                                                                                                     |
| Called functions       | NA                                                                                                                                                                                                                                                                     |

### mem\_address\_width

EEPROM memory address width

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit address width

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit address width

### Example:

```
i2c_memory_write_dma(&hi2c, 0xA0, 0x05, tx_buf, 8, 0xFFFFFFFF);
```

## 5.12.65 i2c\_memory\_read function

The table below describes the function i2c\_memory\_write\_dma

**Table 339. i2c\_memory\_write\_dma function**

| Name                   | Description                                                                                                                                                                                          |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_memory_read                                                                                                                                                                                      |
| Function prototype     | i2c_status_type i2c_memory_read(i2c_handle_type* hi2c,<br>i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t<br>mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout); |
| Function description   | Read EEPROM (through DMA mode). This function is a blocking type. In other words, after the function execution is done, so is data transfer. It is mainly used for PEC transmission and reception.   |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                          |
| Input parameter 2      | mem_address_width: EEPROM memory address width<br>Refer to the “mem_address_width” 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 be read                                                                                                                                                              |
| Input parameter 6      | size: number of data to be read                                                                                                                                                                      |
| Input parameter 7      | Timeout: wait timeout                                                                                                                                                                                |
| Output parameter       | NA                                                                                                                                                                                                   |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                         |
| Required preconditions | NA                                                                                                                                                                                                   |
| Called functions       | NA                                                                                                                                                                                                   |

### mem\_address\_width

EEPROM memory address width

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit address width

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit address width

### Example:

```
i2c_memory_read(&hi2c, 0xA0, 0x05, rx_buf, 8, 0xFFFFFFFF);
```

## 5.12.66 i2c\_memory\_read\_int function

The table below describes the function i2c\_memory\_read\_int

**Table 340. i2c\_memory\_write\_dma function**

| Name                   | Description                                                                                                                                                                                                                                                                 |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | i2c_memory_read_int                                                                                                                                                                                                                                                         |
| Function prototype     | i2c_status_type i2c_memory_read_int(i2c_handle_type* hi2c,<br>i2c_mem_address_width_type mem_address_width, uint16_t address, uint16_t<br>mem_address, uint8_t* pdata, uint16_t size, uint32_t timeout);                                                                    |
| Function description   | Read EEPROM (through interrupt mode). This function is a non-blocking type. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                                 |
| Input parameter 2      | mem_address_width: EEPROM memory address width<br>Refer to the “mem_address_width” 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 be read                                                                                                                                                                                                                                     |
| Input parameter 6      | size: number of data to be read                                                                                                                                                                                                                                             |
| Input parameter 7      | Timeout: wait timeout                                                                                                                                                                                                                                                       |
| Output parameter       | NA                                                                                                                                                                                                                                                                          |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                                |
| Required preconditions | NA                                                                                                                                                                                                                                                                          |
| Called functions       | NA                                                                                                                                                                                                                                                                          |

### mem\_address\_width

EEPROM memory address width

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit address width

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit address width

### Example:

```
i2c_memory_read_int(&hi2c, 0xA0, 0x05, rx_buf, 8, 0xFFFFFFFF);
```

## 5.12.67 i2c\_memory\_read\_dma function

The table below describes the function i2c\_memory\_read\_dma

**Table 341. i2c\_memory\_write\_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 (through DMA mode). This function is a non-blocking type. In other words, after the function is executed, the I <sup>2</sup> C transfer has not completed yet. So in this case, it is possible to call the i2c_wait_end() to wait for the end of transfer |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .                                                                                                                                                                                           |
| Input parameter 2      | mem_address_width: EEPROM memory address width<br>Refer to the “mem_address_width” 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 be read                                                                                                                                                                                                                               |
| Input parameter 6      | size: number of data to be read                                                                                                                                                                                                                                       |
| Input parameter 7      | Timeout: wait timeout                                                                                                                                                                                                                                                 |
| Output parameter       | NA                                                                                                                                                                                                                                                                    |
| Return value           | i2c_status_type: error code<br>Refer to 5.12.42 for details.                                                                                                                                                                                                          |
| Required preconditions | NA                                                                                                                                                                                                                                                                    |
| Called functions       | NA                                                                                                                                                                                                                                                                    |

### mem\_address\_width

EEPROM memory address width

I2C\_MEM\_ADDR\_WIDIH\_8: 8-bit address width

I2C\_MEM\_ADDR\_WIDIH\_16: 16-bit address width

### Example:

```
i2c_memory_read_dma(&hi2c, 0xA0, 0x05, rx_buf, 8, 0xFFFFFFFF);
```

## 5.12.68 i2c\_evt\_irq\_handler function

The table below describes the function i2c\_evt\_irq\_handler

**Table 342. 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. It is used to handle I <sup>2</sup> C event interrupt |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .     |
| Output parameter       | NA                                                                              |
| Return value           | NA                                                                              |
| Required preconditions | NA                                                                              |
| Called functions       | NA                                                                              |

**Example:**

```
void I2C1_EVT_IRQHandler(void)
{
 i2c_evt_irq_handler(&hi2c);
}
```

## 5.12.69 i2c\_err\_irq\_handler function

The table below describes the function i2c\_err\_irq\_handler

**Table 343. 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. It is used to handle I <sup>2</sup> C error interrupt |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .     |
| Output parameter       | NA                                                                              |
| Return value           | NA                                                                              |
| Required preconditions | NA                                                                              |
| Called functions       | NA                                                                              |

**Example:**

```
void I2C1_ERR_IRQHandler(void)
{
 i2c_err_irq_handler(&hi2c);
}
```

## 5.12.70 i2c\_dma\_tx\_irq\_handler function

The table below describes the function i2c\_dma\_tx\_irq\_handler

**Table 344. 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. It is used to handle DMA transmit interrupt |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> .  |
| Output parameter       | NA                                                                           |
| Return value           | NA                                                                           |
| Required preconditions | NA                                                                           |
| Called functions       | NA                                                                           |

**Example:**

```
void DMA1_Channel6_IRQHandler(void)
{
 i2c_dma_rx_irq_handler(&hi2c);
}
```

## 5.12.71 i2c\_dma\_rx\_irq\_handler function

The table below describes the function i2c\_dma\_rx\_irq\_handler

**Table 345. 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. It is used to handle DMA receive interrupt  |
| Input parameter 1      | hi2c: i2c_handle_type pointer<br>Refer to <a href="#">i2c_handle_type</a> . |
| 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 346. Summary of NVIC registers**

| Register | Description                         |
|----------|-------------------------------------|
| iser     | 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 347. 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 348. 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 349. 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: interrupt vector selection<br>Refer to the " <a href="#">irqn</a> " descriptions below for details.                                                |
| Input parameter 2      | preempt_priority: set preemption priority<br>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<br>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()<br>NVIC_EnableIRQ()                                                                                                                   |

#### irqn

irqn is used to select interrupt vectors, including:

|                    |                                   |
|--------------------|-----------------------------------|
| WWDT IRQn:         | Window timer interrupt            |
| PVM IRQn:          | PVM interrupt linked to EXINT     |
| .....              |                                   |
| I2C2_ERR IRQn:     | I <sup>2</sup> C1 error interrupt |
| TMR13_GLOBAL IRQn: | I <sup>2</sup> C2 error interrupt |

**Example:**

```
/* enable nvic irq */
nvic_irq_enable(ADC1_IRQn, 0, 0);
```

### 5.13.3 nvic\_irq\_disable function

The table below describes the function nvic\_irq\_disable.

**Table 350. 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 <a href="#">irqn</a> for details. |
| Output parameter       | NA                                                                        |
| Return value           | NA                                                                        |
| Required preconditions | NA                                                                        |
| Called functions       | NVIC_DisableIRQ()                                                         |

**Example:**

```
/* disable nvic irq */
nvic_irq_disable(ADC1_IRQn);
```

### 5.13.4 nvic\_priority\_group\_config function

The table below describes the function nvic\_priority\_group\_config.

**Table 351. 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<br>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 from the parameters below

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 352. 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<br>The base address can be set in RAM or FLASH.                                                                      |
| Input parameter 2      | Offset: offset address of interrupt vector table<br>This parameter defines the start address of interrupt vector table, so it must be set to a multiple of 0x200. |
| Output parameter       | NA                                                                                                                                                                |
| Return value           | NA                                                                                                                                                                |
| Required preconditions | NA                                                                                                                                                                |
| Called functions       | NA                                                                                                                                                                |

**base**

base is used to select the 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 353. 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,<br>confirm_state new_state)                      |
| Function description   | Configure NVIC low-power mode                                                                                    |
| Input parameter 1      | lp_mode: select low-power modes<br>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<br>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 “at32f425\_pwc.h”:

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

} pwc_type;
```

The table below gives a list of the PWC registers

**Table 354. Summary of PWC registers**

| Register | Description                   |
|----------|-------------------------------|
| ctrl     | Power control register        |
| ctrlsts  | Power control/status register |
| ctrl2    | Power control register 2      |

The table below gives a list of PWC library functions.

**Table 355. 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_flag_clear                 | Clear flag                                        |
| pwc_flag_get                   | 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 356. 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\_battery\_powered\_domain\_access function

The table below describes the function pwc\_battery\_powered\_domain\_access.

**Table 357. pwc\_battery\_powered\_domain\_access function**

| Name                   | Description                                                                                                      |
|------------------------|------------------------------------------------------------------------------------------------------------------|
| Function name          | pwc_battery_powered_domain_access                                                                                |
| Function prototype     | void pwc_battery_powered_domain_access(confirm_state new_state)                                                  |
| Function description   | Battery powered domain access enable                                                                             |
| Input parameter        | new_state: indicates the pre-configured status of battery powered domain<br>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_battery_powered_domain_access(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 358. 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<br>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 a PVM threshold from the optional parameters below:

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 359. 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<br>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 360. 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 pin<br>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<br>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)

PWC\_WAKEUP\_PIN\_2: Standby wakeup pin 1 (corresponding GPIO is PC13)

PWC\_WAKEUP\_PIN\_4: Standby wakeup pin 4 (corresponding GPIO is PA2)

PWC\_WAKEUP\_PIN\_5: Standby wakeup pin 5 (corresponding GPIO is PC5)

PWC\_WAKEUP\_PIN\_6: Standby wakeup pin 6 (corresponding GPIO is PB5)

PWC\_WAKEUP\_PIN\_7: Standby wakeup pin 7 (corresponding GPIO is PB15)

### 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 361. 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: to-be-cleared flag<br>Refer to the <a href="#">pwc_flag</a> description below for details. |
| Output parameter       | NA                                                                                                   |
| Return value           | NA                                                                                                   |
| Required preconditions | NA                                                                                                   |
| Called functions       | NA                                                                                                   |

### pwc\_flag

pwc\_flag is used to select a flag from the optional parameters below:

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 362. 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<br>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 363. 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<br>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 from the optional parameters below:

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 364. 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<br>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\_voltage\_regulate\_set function

The table below describes the function pwc\_voltage\_regulate\_set.

**Table 365. pwc\_voltage\_regulate\_set function**

| Name                   | Description                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------------------|
| Function name          | pwc_voltage_regulate_set                                                                                               |
| Function prototype     | void pwc_voltage_regulate_set(pwc_regulator_type pwc_regulator)                                                        |
| Function description   | Select the status of voltage regulator in Deepsleep mode                                                               |
| Input parameter        | pwc_regulator: select voltage regulator status<br>This parameter can be any enumerated value in the pwc_regulator_type |
| Output parameter       | NA                                                                                                                     |
| Return value           | NA                                                                                                                     |
| Required preconditions | NA                                                                                                                     |
| Called functions       | NA                                                                                                                     |

### **pwc\_regulator**

pwc\_regulator is used to select the status of voltage regulator from the optional parameters below:

PWC\_REGULATOR\_ON: Voltage regulator ON in Deepsleep mode

PWC\_REGULATOR\_LOW\_POWER: Voltage regulator low-power mode in Deepsleep mode

#### **Example:**

```
/* config the voltage regulator mode */
pwc_voltage_regulate_set(PWC_REGULATOR_LOW_POWER);
```

### 5.14.11 pwc\_standby\_mode\_enter function

The table below describes the function pwc\_standby\_mode\_enter

**Table 366. 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 Serial peripheral interface (SPI)/ I<sup>2</sup>S

The SPI register structure spi\_type is defined in the “at32f425\_spi.h”:

```
/*
 * @brief type define spi register all
 */
typedef struct
{
 ...
} spi_type;
```

The table below gives a list of the SPI registers

**Table 367. 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 368. 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_ti_mode_enable                 | SPI TI mode enable                                                     |
| 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         |

### 5.15.1 spi\_i2s\_reset function

The table below describes the function spi\_i2s\_reset.

**Table 369. 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<br>This parameter can be SPI1, SPI2 or SPI3. |
| Output parameter       | NA                                                                         |
| Return value           | NA                                                                         |
| Required preconditions | NA                                                                         |
| Called functions       | crm_periph_reset();                                                        |

**Example:**

```
spi_i2s_reset (SPI1);
```

### 5.15.2 spi\_default\_para\_init function

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

**Table 370. 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: <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                                                                            |

**Example:**

```
spi_init_type spi_init_struct;
spi_default_para_init (&spi_init_struct);
```

### 5.15.3 spi\_init function

The table below describes the function spi\_init.

Table 371. 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<br>This parameter can be SPI1, SPI2 or SPI3.    |
| 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 at32f425\_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);
```

## 5.15.4 spi\_ti\_mode\_enable function

The table below describes the function spi\_crc\_next\_transmit.

**Table 372. spi\_ti\_mode\_enable function**

| Name                   | Description                                                                |
|------------------------|----------------------------------------------------------------------------|
| Function name          | spi_ti_mode_enable                                                         |
| Function prototype     | void spi_ti_mode_enable(spi_type* spi_x, confirm_state new_state);         |
| Function description   | Enable SPI TI mode                                                         |
| Input parameter 1      | spi_x: select SPI peripherals<br>This parameter can be SPI1, SPI2 or SPI3. |
| Input parameter 2      | new_state: enabled or disabled<br>This parameter can be FALSE or TRUE.     |
| Output parameter       | NA                                                                         |
| Return value           | NA                                                                         |
| Required preconditions | NA                                                                         |
| Called functions       | NA                                                                         |

**Example:**

```
/* spi ti mode enable */
spi_ti_mode_enable (SPI1, TRUE);
```

## 5.15.5 spi\_crc\_next\_transmit function

The table below describes the function spi\_crc\_next\_transmit.

**Table 373. 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<br>This parameter can be SPI1, SPI2 or SPI3. |
| Output parameter       | NA                                                                         |
| Return value           | NA                                                                         |
| Required preconditions | NA                                                                         |
| Called functions       | NA                                                                         |

**Example:**

```
spi_crc_next_transmit (SPI1);
```

## 5.15.6 spi\_crc\_polynomial\_set function

The table below describes the function spi\_crc\_polynomial\_set.

**Table 374. 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<br>This parameter can be SPI1, SPI2 or SPI3. |
| Input parameter 2      | crc_poly: CRC polynomial<br>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);
```

## 5.15.7 spi\_crc\_polynomial\_get function

The table below describes the function spi\_crc\_polynomial\_get.

**Table 375. 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<br>This parameter can be SPI1, SPI2 or SPI3. |
| Output parameter       | NA                                                                        |
| Return value           | CRC polynomial<br>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);
```

## 5.15.8 spi\_crc\_enable function

The table below describes the function spi\_crc\_enable.

**Table 376. 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<br>This parameter can be SPI1, SPI2 or SPI3. |
| Input parameter 2      | new_state: enabled or disabled<br>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);
```

## 5.15.9 spi\_crc\_value\_get function

The table below describes the function spi\_crc\_value\_get.

**Table 377. 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<br>This parameter can be SPI1, SPI2 or SPI3.          |
| 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);
```

### 5.15.10 spi\_hardware\_cs\_output\_enable function

The table below describes the function spi\_hardware\_cs\_output\_enable.

**Table 378. 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<br>This parameter can be SPI1, SPI2 or SPI3.     |
| Input parameter 2      | new_state: enabled or disabled<br>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);
```

### 5.15.11 spi\_software\_cs\_internal\_level\_set function

The table below describes the function spi\_software\_cs\_internal\_level\_set.

**Table 379. 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<br>This parameter can be SPI1, SPI2 or SPI3.                                                                   |
| 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;<br>2. In master mode, the "level" value must be<br>"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);
```

### 5.15.12 spi\_frame\_bit\_num\_set function

The table below describes the function spi\_frame\_bit\_num\_set.

**Table 380. 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 software CS internal level                                               |
| Input parameter 1      | spi_x: select SPI peripheral<br>This parameter can be SPI1, SPI2 or SPI3.    |
| 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);
```

### 5.15.13 spi\_half\_duplex\_direction\_set function

The table below describes the function spi\_half\_duplex\_direction\_set.

**Table 381. 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<br>This parameter can be SPI1, SPI2 or SPI3.                      |
| 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);
```

## 5.15.14 spi\_enable function

The table below describes the function spi\_enable.

**Table 382. 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<br>This parameter can be SPI1, SPI2 or SPI3. |
| Input parameter 2      | new_state: enabled or disabled<br>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);
```

## 5.15.15 i2s\_default\_para\_init function

The table below describes the function i2s\_default\_para\_init.

**Table 383. 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);
```

## 5.15.16 i2s\_init function

The table below describes the function i2s\_init.

Table 384. 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<br>This parameter can be SPI1, SPI2 or SPI3. |
| 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 at32f425\_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:  | I <sup>2</sup> S slave transmit  |
| I2S_MODE_SLAVE_RX:  | I <sup>2</sup> S slave receive   |
| I2S_MODE_MASTER_TX: | I <sup>2</sup> S master transmit |
| I2S_MODE_MASTER_RX: | I <sup>2</sup> S 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:      | I <sup>2</sup> S sampling frequency 8K      |
| I2S_AUDIO_FREQUENCY_11_025K: | I <sup>2</sup> S sampling frequency 11.025K |
| I2S_AUDIO_FREQUENCY_16K:     | I <sup>2</sup> S sampling frequency 16K     |
| I2S_AUDIO_FREQUENCY_22_05K:  | I <sup>2</sup> S sampling frequency 22.05K  |
| I2S_AUDIO_FREQUENCY_32K:     | I <sup>2</sup> S sampling frequency 32K     |

|                            |                                           |
|----------------------------|-------------------------------------------|
| I2S_AUDIO_FREQUENCY_44_1K: | I <sup>2</sup> S sampling frequency 44.1K |
| I2S_AUDIO_FREQUENCY_48K:   | I <sup>2</sup> S sampling frequency 48K   |
| I2S_AUDIO_FREQUENCY_96K:   | I <sup>2</sup> S sampling frequency 96K   |
| I2S_AUDIO_FREQUENCY_192K:  | I <sup>2</sup> S 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);
```

### 5.15.17 i2s\_enable function

The table below describes the function i2s\_enable.

Table 385. 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<br>This parameter can be SPI1, SPI2 or SPI3. |
| Input parameter 2      | new_state: Enable or disable<br>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);
```

## 5.15.18 spi\_i2s\_interrupt\_enable function

The table below describes the function spi\_i2s\_interrupt\_enable.

**Table 386. 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<br>This parameter can be SPI1, SPI2 or SPI3.                      |
| Input parameter 2      | <i>spi_i2s_int</i> : select SPI interrupts                                                     |
| Input parameter 3      | new_state: Enable or disable<br>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 interrupt selection.

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);
```

## 5.15.19 spi\_i2s\_dma\_transmitter\_enable function

The table below describes the function spi\_i2s\_dma\_transmitter\_enable.

**Table 387. 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<br>This parameter can be SPI1, SPI2 or SPI3.      |
| Input parameter 2      | new_state: enabled or disabled<br>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);
```

**5.15.20 spi\_i2s\_dma\_receiver\_enable function**

The table below describes the function `spi_i2s_dma_receiver_enable`.

**Table 388. `spi_i2s_dma_receiver_enable` function**

| Name                   | Description                                                                              |
|------------------------|------------------------------------------------------------------------------------------|
| Function name          | <code>spi_i2s_dma_receiver_enable</code>                                                 |
| Function prototype     | <code>void spi_i2s_dma_receiver_enable(spi_type* spi_x, confirm_state new_state);</code> |
| Function description   | Enable SPI/I <sup>2</sup> S DMA receiver                                                 |
| Input parameter 1      | <code>spi_x</code> : select SPI peripheral<br>This parameter can be SPI1, SPI2 or SPI3.  |
| Input parameter 2      | <code>new_state</code> : enabled or disabled<br>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);
```

**5.15.21 spi\_i2s\_data\_transmit function**

The table below describes the function `spi_i2s_data_transmit`.

**Table 389. `spi_i2s_data_transmit` function**

| Name                   | Description                                                                                                                                    |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | <code>spi_i2s_data_transmit</code>                                                                                                             |
| Function prototype     | <code>void spi_i2s_data_transmit(spi_type* spi_x, uint16_t tx_data);</code>                                                                    |
| Function description   | SPI/I <sup>2</sup> S sends data                                                                                                                |
| Input parameter 1      | <code>spi_x</code> : select SPI peripheral<br>This parameter can be SPI1, SPI2 or SPI3.                                                        |
| Input parameter 2      | <code>tx_data</code> : data to send<br>Value range (for 8-bit bit in a frame): 0x00~0xFF<br>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);
```

## 5.15.22 spi\_i2s\_data\_receive function

The table below describes the function spi\_i2s\_data\_receive.

**Table 390. 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<br>This parameter can be SPI1, SPI2 or SPI3.                                                           |
| Output parameter       | rx_data: data to receive<br>Value range (for 8-bit bit in a frame): 0x00~0xFF<br>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);
```

## 5.15.23 spi\_i2s\_flag\_get function

The table below describes the function spi\_i2s\_flag\_get.

**Table 391. 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<br>This parameter can be SPI1, SPI2 or SPI3.                                   |
| Input parameter 2      | <a href="#">spi_i2s_flag</a> : flag selection<br>Refer to the “spi_i2s_flag” description below for details. |
| Output parameter       | NA                                                                                                          |
| Return value           | flag_status: flag status<br>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 a flag from the optional parameters below:

- 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: I<sup>2</sup>S audio channel state (indicating left/right channel)
- 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 |
| SPI_I2S_BF_FLAG:    | SPI/I <sup>2</sup> S busy                   |
| SPI_CSPAS_FLAG:     | SPI CS pulse error                          |

**Example:**

```
/* get receive data buffer full flag */
flag_status status;
status = spi_i2s_flag_get(SPI, SPI_I2S_RDBF_FLAG);
```

### 5.15.24 spi\_i2s\_interrupt\_flag\_get function

The table below describes the function spi\_i2s\_interrupt\_flag\_get.

**Table 392. 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   | Read the selected SPI/I <sup>2</sup> S interrupt flag                                                   |
| Input parameter 1      | spi_x: select SPI peripheral<br>This parameter can be SPI1, SPI2 or SPI3.                               |
| Input parameter 2      | <i>spi_i2s_flag</i> : select a flag to clear<br>Refer to the “ <i>spi_i2s_flag</i> ” below for details. |
| Output parameter       | NA                                                                                                      |
| Return value           | flag_status: SET or RESET.                                                                              |
| 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   |
| 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     |
| SPI_CSPAS_FLAG:     | SPI CS pulse error                              |

**Example:**

```
/* get receive data buffer full flag */
flag_status status;
status = spi_i2s_interrupt_flag_get(SPI, SPI_I2S_RDBF_FLAG);
```

## 5.15.25 spi\_i2s\_flag\_clear function

The table below describes the function spi\_i2s\_flag\_clear.

Table 393. 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<br>This parameter can be SPI1, SPI2 or SPI3.                                  |
| Input parameter 2      | <i>spi_i2s_flag</i> : select a flag to clear<br>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:     | 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   |
| SPI_CSPAS_FLAG:     | SPI CS pulse 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 (SPI, SPI_I2S_RDBF_FLAG);
```

## 5.16 System configuration controller (SCFG)

The SCFG register structure scfg\_type is defined in the “at32f425\_scfg.h”

```
/*
 * @brief type define scfg register all
 */
typedef struct
{
 ...
} scfg_type;
```

The table below gives a list of the SCFG registers

**Table 394. Summary of SCFG registers**

| Register     | Description                                      |
|--------------|--------------------------------------------------|
| scfg_cfg1    | SCFG configuration register 1                    |
| scfg_cfg2    | SCFG configuration register 2                    |
| scfg_exintc1 | SCFG external interrupt configuration register 1 |
| scfg_exintc2 | SCFG external interrupt configuration register 2 |
| scfg_exintc3 | SCFG external interrupt configuration register 3 |
| scfg_exintc4 | SCFG external interrupt configuration register 4 |
| scfg_uhdrv   | SCFG ultra-high drive capability register        |

The table below gives a list of SCFG library functions.

**Table 395. Summary of SCFG library functions**

| Function name                 | Description                                      |
|-------------------------------|--------------------------------------------------|
| scfg_reset                    | SCFG reset                                       |
| scfg_infrared_config          | infrared configuration                           |
| scfg_mem_map_get              | Get memory address map                           |
| scfg_pa11pa12_pin_remap       | Remap PA11 and PA12                              |
| scfg_exint_line_config        | External interrupt line configuration            |
| scfg_pins_ultra_driven_enable | Pin ultra-high current sinking capability enable |
| scfg_i2s_full_duplex_config   | I2S full-duplex mode configuration               |
| scfg_pvm_lock_enable          | PVM Lock enable                                  |

## 5.16.1 scfg\_reset function

The table below describes the function scfg\_reset.

**Table 396. scfg\_reset function**

| Name                   | Description            |
|------------------------|------------------------|
| Function name          | scfg_reset             |
| Function prototype     | void scfg_reset(void); |
| Function description   | Reset SCFG             |
| Input parameter        | NA                     |
| Output parameter       | NA                     |
| Return value           | NA                     |
| Required preconditions | NA                     |
| Called functions       | NA                     |

**Example:**

```
scfg_reset();
```

## 5.16.2 scfg\_infrared\_config function

The table below describes the function scfg\_infrared\_config.

**Table 397. scfg\_infrared\_config function**

| Name                   | Description                                                                            |
|------------------------|----------------------------------------------------------------------------------------|
| Function name          | scfg_infrared_config                                                                   |
| Function prototype     | void scfg_infrared_config(scfg_ir_source_type source, scfg_ir_polarity_type polarity); |
| Function description   | Infrared configuration                                                                 |
| Input parameter 1      | Source: infrared modulation signal source                                              |
| Input parameter 2      | Polarity: output signal polarity                                                       |
| Output parameter       | NA                                                                                     |
| Return value           | NA                                                                                     |
| Required preconditions | NA                                                                                     |
| Called functions       | NA                                                                                     |

**scfg\_ir\_source\_type**

Select infrared signal source.

SCFG\_IR\_SOURCE\_TMR16: Infrared signal source is TMR16

**scfg\_ir\_polarity\_type**

Select infrared signal polarity.

SCFG\_IR\_POLARITY\_NO\_AFFECTE: Infrared output signal not inverted

SCFG\_IR\_POLARITY\_REVERSE: Infrared output signal inverted

**Example:**

```
scfg_infrared_config(SCFG_IR_SOURCE_TMR16, SCFG_IR_POLARITY_NO_AFFECTE);
```

### 5.16.3 scfg\_mem\_map\_get function

The table below describes the function scfg\_mem\_map\_get.

**Table 398. scfg\_mem\_map\_get function**

| Name                   | Description                                                        |
|------------------------|--------------------------------------------------------------------|
| Function name          | scfg_mem_map_get                                                   |
| Function prototype     | uint8_t scfg_mem_map_get(void);                                    |
| Function description   | Get the status of a memory being mapped on the address 0x00000000. |
| Input parameter        | NA                                                                 |
| Output parameter       | NA                                                                 |
| Return value           | uint8_t: memory address map type                                   |
| Required preconditions | NA                                                                 |
| Called functions       | NA                                                                 |

#### Memory address map type

SCFG\_MEM\_MAP\_MAIN\_MEMORY: Main memory is mapped to 0x00000000

SCFG\_MEM\_MAP\_BOOT\_MEMORY: Boot memory is mapped to 0x00000000

SCFG\_MEM\_MAP\_INTERNAL\_SRAM: Internal memory is mapped to 0x00000000

#### Example:

```
uint8_t value;
value = scfg_mem_map_get();
```

### 5.16.4 scfg\_pa11pa12\_pin\_remap function

The table below describes the function scfg\_pa11pa12\_pin\_remap.

**Table 399. scfg\_pa11pa12\_pin\_remap function**

| Name                   | Description                                                       |
|------------------------|-------------------------------------------------------------------|
| Function name          | scfg_pa11pa12_pin_remap                                           |
| Function prototype     | void scfg_pa11pa12_pin_remap(scfg_pa11pa12_remap_type pin_remap); |
| Function description   | Remap PA11 and PA12                                               |
| Input parameter        | pin_remap: remap selection                                        |
| Output parameter       | NA                                                                |
| Return value           | NA                                                                |
| Required preconditions | NA                                                                |
| Called functions       | NA                                                                |

#### scfg\_pa11pa12\_remap\_type

Remap selection.

SCFG\_PA11PA12\_NO\_REMAP: PA11 and PA12 are not remapped

SCFG\_PA11PA12\_TO\_PA9PA10: PA11 and PA12 are remapped on to PA9 and PA10

#### Example:

```
scfg_pa11pa12_pin_remap(SCFG_PA11PA12_TO_PA9PA10);
```

## 5.16.5 scfg\_exint\_line\_config function

The table below describes the function scfg\_exint\_line\_config.

Table 400. scfg\_exint\_line\_config function

| Name                   | Description                                                                                          |
|------------------------|------------------------------------------------------------------------------------------------------|
| Function name          | scfg_exint_line_config                                                                               |
| Function prototype     | void scfg_exint_line_config(scfg_port_source_type port_source,<br>scfg_pins_source_type pin_source); |
| Function description   | External interrupt line configuration                                                                |
| Input parameter 1      | port_source: port source                                                                             |
| Input parameter 2      | pin_source: pin source                                                                               |
| Output parameter       | NA                                                                                                   |
| Return value           | NA                                                                                                   |
| Required preconditions | NA                                                                                                   |
| Called functions       | NA                                                                                                   |

### scfg\_port\_source\_type

SCFG\_PORT\_SOURCE\_GPIOA: port A

SCFG\_PORT\_SOURCE\_GPIOB: port B

SCFG\_PORT\_SOURCE\_GPIOC: port C

SCFG\_PORT\_SOURCE\_GPIOD: port D

SCFG\_PORT\_SOURCE\_GPIOE: port E

SCFG\_PORT\_SOURCE\_GPIOF: port F

### scfg\_pins\_source\_type

SCFG\_PINS\_SOURCE0: pin 0

SCFG\_PINS\_SOURCE1: pin 1

SCFG\_PINS\_SOURCE2: pin 2

.....

SCFG\_PINS\_SOURCE13: pin 13

SCFG\_PINS\_SOURCE14: pin 14

SCFG\_PINS\_SOURCE15: pin15

### Example:

```
scfg_exint_line_config(SCFG_PORT_SOURCE_GPIOA, SCFG_PINS_SOURCE1);
```

## 5.16.6 scfg\_pins\_ultra\_driven\_enable function

The table below describes the function scfg\_pins\_ultra\_driven\_enable.

**Table 401. scfg\_pins\_ultra\_driven\_enable function**

| Name                   | Description                                                                                     |
|------------------------|-------------------------------------------------------------------------------------------------|
| Function name          | scfg_pins_ultra_driven_enable                                                                   |
| Function prototype     | void scfg_pins_ultra_driven_enable(scfg_ultra_driven_pins_type value, confirm_state new_state); |
| Function description   | Enable pin ultra-high current sinking capability.                                               |
| Input parameter 1      | Value: pin                                                                                      |
| Input parameter 2      | new_state: interrupt status<br>This parameter can be TRUE or FALSE.                             |
| Output parameter       | NA                                                                                              |
| Return value           | NA                                                                                              |
| Required preconditions | NA                                                                                              |
| Called functions       | NA                                                                                              |

### scfg\_ultra\_driven\_pins\_type

SCFG\_ULTRA\_DRIVEN\_PB8: PB8  
 SCFG\_ULTRA\_DRIVEN\_PB9: PB9  
 SCFG\_ULTRA\_DRIVEN\_PB13: PB13  
 SCFG\_ULTRA\_DRIVEN\_PB14: PB14

#### Example:

```
scfg_pins_ultra_driven_enable(SCFG_ULTRA_DRIVEN_PB8, TRUE);
```

## 5.16.7 scfg\_i2s\_full\_duplex\_config function

The table below describes the function scfg\_i2s\_full\_duplex\_config.

**Table 402. scfg\_i2s\_full\_duplex\_config function**

| Name                   | Description                                                      |
|------------------------|------------------------------------------------------------------|
| Function name          | scfg_i2s_full_duplex_config                                      |
| Function prototype     | void scfg_i2s_full_duplex_config(scfg_i2s_type i2s_full_duplex); |
| Function description   | I <sup>2</sup> S full-duplex mode configuration                  |
| Input parameter        | i2s_full_duplex: full-duplex mode selection                      |
| Output parameter       | NA                                                               |
| Return value           | NA                                                               |
| Required preconditions | NA                                                               |
| Called functions       | NA                                                               |

### scfg\_i2s\_type

Set full-duplex mode.

SCFG\_FULL\_DUPLEX\_I2S\_NONE: None  
 SCFG\_FULL\_DUPLEX\_I2S1\_I2S3: Combine I<sup>2</sup>S1 and I<sup>2</sup>S3 as full-duplex mode  
 SCFG\_FULL\_DUPLEX\_I2S2\_I2S3: Combine I<sup>2</sup>S2 and I<sup>2</sup>S3 as full-duplex mode  
 SCFG\_FULL\_DUPLEX\_I2S1\_I2S2: Combine I<sup>2</sup>S1 and I<sup>2</sup>S2 as full-duplex mode

#### Example:

```
scfg_i2s_full_duplex_config(SCFG_FULL_DUPLEX_I2S1_I2S3);
```

## 5.16.8 scfg\_pvm\_lock\_enable function

The table below describes the function scfg\_adc\_dma\_channel\_remap.

**Table 403. scfg\_adc\_dma\_channel\_remap function**

| Name                   | Description                                                            |
|------------------------|------------------------------------------------------------------------|
| Function name          | scfg_pvm_lock_enable                                                   |
| Function prototype     | void scfg_pvm_lock_enable(confirm_state new_state);                    |
| Function description   | PVM Lock enable                                                        |
| Input parameter        | new_state: enabled or disabled<br>This parameter can be FALSE or TRUE. |
| Output parameter       | NA                                                                     |
| Return value           | NA                                                                     |
| Required preconditions | NA                                                                     |
| Called functions       | NA                                                                     |

**Example:**

```
scfg_pvm_lock_enable(TRUE);
```

## 5.17 SysTick

The SysTick register structure SysTick\_Type is defined in the “core\_cm4.h”:

```
typedef struct
```

```
{
```

```
...
```

```
}
```

The table below gives a list of the SysTick registers

**Table 404. 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 405. 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 406. 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 407. 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 enable interrupt            |
| 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 TMR

The TMR register structure tmr\_type is defined in the "at32f425\_tmr.h":

```
/*
 * @brief type define tmr register all
 */
typedef struct
{
 } tmr_type;
```

The table below gives a list of the TMR registers

**Table 408. 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              |
| rmp      | TMR channel input remap register   |

The table below gives a list of TMR library functions.

Table 409. 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_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_interrupt_flag_get             | Get TMR interrupt flag status                   |
| 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 |
| tmr_iremap_config               | Set TMR internal remap           |

### 5.18.1 tmr\_reset function

The table below describes the function tmr\_reset.

Table 410. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 and TMR17. |
| Output parameter       | NA                                                                                                           |
| Return value           | NA                                                                                                           |
| Required preconditions | NA                                                                                                           |
| Called functions       | crm_periph_reset();                                                                                          |

**Example:**

```
tmr_reset(TMR1);
```

### 5.18.2 tmr\_counter\_enable function

The table below describes the function tmr\_counter\_enable.

Table 411. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 and TMR17. |
| 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);
```

### 5.18.3 tmr\_output\_default\_para\_init function

The table below describes the function tmr\_output\_default\_para\_init.

**Table 412. 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 413. 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);
```

### 5.18.4 tmr\_input\_default\_para\_init function

The table below describes the function tmr\_input\_default\_para\_init.

**Table 414. 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 415. 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);
```

**5.18.5 tmr\_brkdt\_default\_para\_init function**

The table below describes the function tmr\_brkdt\_default\_para\_init.

**Table 416. 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 417. 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);
```

## 5.18.6 tmr\_base\_init function

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

**Table 418. 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 peripherals, including: TMR1, TMR2, TMR3, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 and TMR17.    |
| Input parameter 2      | tmr_pr: timer period value, 0x0000~0xFFFF for 16-bit timer, and 0x0000_0000~0xFFFF_FFFF for 32-bit timer. |
| 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);
```

## 5.18.7 tmr\_clock\_source\_div\_set function

The table below describes the function tmr\_clock\_source\_div\_set.

**Table 419. 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                                                              |
| Input parameter 1      | tmr_x: TMR peripherals, including: TMR1, TMR2, TMR3, TMR13, TMR14, TMR15, TMR16 and TMR17. |
| 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);
```

## 5.18.8 tmr\_cnt\_dir\_set function

The table below describes the function tmr\_cnt\_dir\_set.

**Table 420. 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 and TMR3. |
| 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);
```

## 5.18.9 tmr\_repetition\_counter\_set function

The table below describes the function tmr\_repetition\_counter\_set.

**Table 421. tmr\_repetition\_counter\_set function**

| Name                   | Description                                                               |
|------------------------|---------------------------------------------------------------------------|
| Function name          | tmr_repetition_counter_set                                                |
| Function prototype     | void tmr_repetition_counter_set(tmr_type *tmr_x, uint16_t tmr_rpr_value); |
| Function description   | Set repetition period register (rpr)                                      |
| Input parameter 1      | tmr_x: TMR peripheral. It includes: TMR1.                                 |
| 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);
```

## 5.18.10 tmr\_counter\_value\_set function

The table below describes the function tmr\_counter\_value\_set.

**Table 422. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| Input parameter 2      | tmr_cnt_value: timer counter value, 0x0000~0xFFFF for 16-bit timer;<br>0x0000_0000~0xFFFF_FFFF for 32-bit timer            |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
tmr_counter_value_set(TMR1, 0xFFFF);
```

## 5.18.11 tmr\_counter\_value\_get function

The table below describes the function tmr\_counter\_value\_get.

**Table 423. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| 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);
```

## 5.18.12 tmr\_div\_value\_set function

The table below describes the function tmr\_div\_value\_set.

**Table 424. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| Input parameter 2      | tmr_div_value: timer frequency division value. 0x0000~0xFFFF for 16-bit timer; 0x0000_0000~0xFFFF_FFFF for 32-bit timer    |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
tmr_div_value_set(TMR1, 0xFFFF);
```

## 5.18.13 tmr\_div\_value\_get function

The table below describes the function tmr\_div\_value\_get.

**Table 425. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| 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);
```

## 5.18.14 tmr\_output\_channel\_config function

The table below describes the function tmr\_output\_channel\_config.

**Table 426. 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, TMR13, TMR14, TMR15, TMR16 or TMR17.                   |
| Input parameter 2      | tmr_channel: timer channel                                                                                                       |
| 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

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 at32f425\_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: CxORAW forced low

TMR\_OUTPUT\_CONTROL\_FORCE\_HIGH: CxORAW forced 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);
```

## 5.18.15 tmr\_output\_channel\_mode\_select function

The table below describes the function tmr\_output\_channel\_mode\_select.

**Table 427. 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, TMR13, TMR14, TMR15, TMR16 or TMR17.                   |
| Input parameter 2      | tmr_channel: refer to the “ <a href="#">tmr_channel</a> ” descriptions below for details                                         |
| Input parameter 3      | oc_mode: refer to the “ <a href="#">oc_mode</a> ” descriptions below for details                                                 |
| 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

### 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: CxORAW forced low

TMR\_OUTPUT\_CONTROL\_FORCE\_HIGH: CxORAW forced 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);
```

## 5.18.16 tmr\_period\_value\_set function

The table below describes the function tmr\_period\_value\_set.

**Table 428. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| Input parameter 2      | tmr_pr_value: timer period value., 0x0000~0xFFFF for 16-bit timer;<br>0x0000_0000~0xFFFF_FFFF for 32-bit timer             |
| Output parameter       | NA                                                                                                                         |
| Return value           | NA                                                                                                                         |
| Required preconditions | NA                                                                                                                         |
| Called functions       | NA                                                                                                                         |

**Example:**

```
tmr_period_value_set(TMR1, 0xFFFF);
```

## 5.18.17 tmr\_period\_value\_get function

The table below describes the function tmr\_period\_value\_get.

**Table 429. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| 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);
```

## 5.18.18 tmr\_channel\_value\_set function

The table below describes the function tmr\_channel\_value\_set.

Table 430. 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, TMR13, TMR14, TMR15, TMR16 or TMR17.   |
| Input parameter 2      | tmr_channel: timer channel                                                                                       |
| Input parameter 3      | tmr_channel_value: timer channel value. 0x0000~0xFFFF for 16-bit timer; 0x0000_0000~0xFFFF_FFFF for 32-bit timer |
| Output parameter       | NA                                                                                                               |
| Return value           | NA                                                                                                               |
| Required preconditions | NA                                                                                                               |
| Called functions       | NA                                                                                                               |

### tmr\_channel

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);
```

## 5.18.19 tmr\_channel\_value\_get function

The table below describes the function tmr\_channel\_value\_get.

**Table 431. 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, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| Input parameter 2      | tmr_channel: timer channel                                                                                     |
| Output parameter       | Timer channel value                                                                                            |
| Return value           | NA                                                                                                             |
| Required preconditions | NA                                                                                                             |
| Called functions       | NA                                                                                                             |

### tmr\_channel

Select a TMR channel.

TMR\_SELECT\_CHANNEL\_1: TMR channel 1

TMR\_SELECT\_CHANNEL\_2: TMR channel 2

TMR\_SELECT\_CHANNEL\_3: TMR channel 3

TMR\_SELECT\_CHANNEL\_4: TMR channel 4

### Example:

```
uint32_t ch_value;
ch_value = tmr_channel_value_get(TMR1, TMR_SELECT_CHANNEL_1);
```

## 5.18.20 tmr\_period\_buffer\_enable function

The table below describes the function tmr\_period\_buffer\_enable.

**Table 432. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| 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);
```

## 5.18.21 tmr\_output\_channel\_buffer\_enable function

The table below describes the function tmr\_output\_channel\_buffer\_enable.

Table 433. 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,<br>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, TMR13, TMR14, TMR15, TMR16 or TMR17.           |
| 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

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_output_channel_buffer_enable(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

## 5.18.22 tmr\_output\_channel\_immediately\_set function

The table below describes the function tmr\_output\_channel\_immediately\_set.

**Table 434. 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,<br>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, TMR13, TMR14, TMR15, TMR16 or TMR17.             |
| Input parameter 2      | tmr_channel: timer channel                                                                                                 |
| Input parameter 3      | new_state: indicates the status of output channel enable.<br>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\_2: Timer channel 2

TMR\_SELECT\_CHANNEL\_3: Timer channel 3

TMR\_SELECT\_CHANNEL\_4: Timer channel 4

### Example:

```
tmr_output_channel_immediately_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

## 5.18.23 tmr\_output\_channel\_switch\_set function

The table below describes the function tmr\_output\_channel\_switch\_set.

**Table 435. 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, TMR13, TMR14, TMR15, TMR16 or TMR17.       |
| Input parameter 2      | tmr_channel: timer channel                                                                                           |
| Input parameter 3      | new_state: indicates the status of output channel switch.<br>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\_2: Timer channel 2

TMR\_SELECT\_CHANNEL\_3: Timer channel 3

TMR\_SELECT\_CHANNEL\_4: Timer channel 4

### Example:

```
tmr_output_channel_switch_set(TMR1, TMR_SELECT_CHANNEL_1, TRUE);
```

## 5.18.24 tmr\_one\_cycle\_mode\_enable function

The table below describes the function tmr\_one\_cycle\_mode\_enable.

**Table 436. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| Input parameter 2      | new_state: indicates the status of one-cycle mode.<br>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);
```

## 5.18.25 tmr\_32\_bit\_function\_enable function

The table below describes the function tmr\_32\_bit\_function\_enable.

**Table 437. 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.                                   |
| Input parameter 2      | new_state: the status of 32-bit mode<br>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);
```

## 5.18.26 tmr\_overflow\_request\_source\_set function

The table below describes the function tmr\_overflow\_request\_source\_set.

**Table 438. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| 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 mode timer controller

TRUE: Counter overflow only.

**Example:**

```
tmr_overflow_request_source_set(TMR1, TRUE);
```

## 5.18.27 tmr\_overflow\_event\_disable function

The table below describes the function tmr\_overflow\_event\_disable.

**Table 439. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| 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 mode timer controller

TRUE: Disable overflow event generation

### Example:

```
tmr_overflow_event_disable(TMR1, TRUE);
```

## 5.18.28 tmr\_input\_channel\_init function

The table below describes the function tmr\_input\_channel\_init.

**Table 440. 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, TMR13, TMR14, TMR15, TMR16 or TMR17.                    |
| 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 at32f425\_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: | Timer channel 1 |
| TMR_SELECT_CHANNEL_2: | Timer channel 2 |
| TMR_SELECT_CHANNEL_3: | Timer channel 3 |
| TMR_SELECT_CHANNEL_4: | Timer 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);
```

## 5.18.29 tmr\_channel\_enable function

The table below describes the function tmr\_channel\_enable.

Table 441. 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, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| Input parameter 2      | tmr_channel: timer channel                                                                                     |
| Input parameter 3      | new_state: indicates the status of timer channels.<br>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);
```

### 5.18.30 tmr\_input\_channel\_filter\_set function

The table below describes the function tmr\_input\_channel\_filter\_set.

**Table 442. 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, TMR13, TMR14, TMR15, TMR16 or TMR17.  |
| Input parameter 2      | tmr_channel: timer channel                                                                                      |
| 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);
```

### 5.18.31 tmr\_pwm\_input\_config function

The table below describes the function tmr\_pwm\_input\_config.

**Table 443. 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 or 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                                                                                                                              |

#### 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);
```

## 5.18.32 tmr\_channel1\_input\_select function

The table below describes the function tmr\_channel1\_input\_select.

**Table 444. tmr\_channel1\_input\_select function**

| Name                   | Description                                                                                        |
|------------------------|----------------------------------------------------------------------------------------------------|
| Function name          | tmr_channel1_input_select                                                                          |
| Function prototype     | void tmr_channel1_input_select(tmr_type *tmr_x,<br>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 or TMR3.                        |
| 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);
```

### 5.18.33 tmr\_input\_channel\_divider\_set function

The table below describes the function tmr\_input\_channel\_divider\_set.

**Table 445. 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, TMR13, TMR14, TMR15, TMR16 or TMR17.                           |
| Input parameter 2      | tmr_channel: timer channel                                                                                                               |
| 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);
```

### 5.18.34 tmr\_primary\_mode\_select function

The table below describes the function tmr\_primary\_mode\_select.

Table 446. 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 primary (master) mode                                                        |
| Input parameter 1      | tmr_x: indicates the selected TMR peripheral. It can be TMR1, TMR2, TMR3, TMR6 or TMR7. |
| Input parameter 2      | primary_mode: master mode                                                               |
| Output parameter       | NA                                                                                      |
| Return value           | NA                                                                                      |
| Required preconditions | NA                                                                                      |
| Called functions       | NA                                                                                      |

#### primary\_mode

Select primary 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 pulse |
| 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);
```

### 5.18.35 tmr\_sub\_mode\_select function

The table below describes the function tmr\_sub\_mode\_select.

**Table 447. 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 or 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_DISABLE:          | 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);
```

### 5.18.36 tmr\_channel\_dma\_select function

The table below describes the function tmr\_channel\_dma\_select.

**Table 448. 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, TMR15, TMR16 or TMR17. |
| 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);
```

### 5.18.37 tmr\_hall\_select function

The table below describes the function tmr\_hall\_select

**Table 449. 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.   |
| 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);
```

### 5.18.38 tmr\_channel\_buffer\_enable function

The table below describes the function tmr\_channel\_buffer\_enable.

**Table 450. 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, TMR15, TMR16 or TMR17.                              |
| Input parameter 2      | new_state: indicates the status of TMR channel buffer.<br>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);
```

### 5.18.39 tmr\_trigger\_input\_select function

The table below describes the function tmr\_trigger\_input\_select.

**Table 451. 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 or 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\_IS0: 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);
```

### 5.18.40 tmr\_sub\_sync\_mode\_set function

The table below describes the function tmr\_sub\_sync\_mode\_set.

**Table 452. 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, it can be TMR1, TMR2, TMR3 or TMR15.                                                            |
| Input parameter 2      | new_state: indicates the status of TMR slave timer synchronization mode<br>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);
```

### 5.18.41 tmr\_dma\_request\_enable function

The table below describes the function tmr\_dma\_request\_enable.

**Table 453. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| Input parameter 2      | dma_request: DMA request                                                                                                   |
| Input parameter 3      | new_state: indicates the status of DMA request.<br>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);
```

### 5.18.42 tmr\_interrupt\_enable function

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

**Table 454. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| Input parameter 2    | tmr_interrupt: TMR interrupts                                                                                              |
| Input parameter 3    | new_state: indicates the status of TMR interrupts.<br>This parameter can be Enable (TRUE) or Disable (FALSE).              |
| Output parameter     | NA                                                                                                                         |
| Return value         | NA                                                                                                                         |

| Name                   | Description |
|------------------------|-------------|
| 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);
```

### 5.18.43 tmr\_interrupt\_flag\_get function

The table below describes the function tmr\_interrupt\_flag\_get

**Table 455. 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 TMR interrupt flag status                                                                                              |
| Input parameter 1      | tmr_x: indicates the selected TMR peripheral. It can be TMR1, TMR2, TMR3, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| Input parameter 2      | tmr_flag: Flag selection<br>Refer to the "tmr_flag" below for details.                                                     |
| Output parameter       | NA                                                                                                                         |
| Return value           | flag_status: indicates the status of flags<br>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)
```

### 5.18.44 tmr\_flag\_get function

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

**Table 456. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| Input parameter 2      | tmr_flag: Flag selection<br>Refer to the “tmr_flag” description below for details.                                         |
| Output parameter       | NA                                                                                                                         |
| Return value           | flag_status: indicates the status of flags<br>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)
```

### 5.18.45 tmr\_flag\_clear function

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

**Table 457. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |

| Name                   | Description                                                                |
|------------------------|----------------------------------------------------------------------------|
| Input parameter 2      | tmr_flag: flag selection<br>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);
```

### 5.18.46 tmr\_event\_sw\_trigger function

The table below describes the function tmr\_event\_sw\_trigger

**Table 458. 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, TMR6, TMR7, TMR13, TMR14, TMR15, TMR16 or TMR17. |
| 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:**

```
tmr_event_sw_trigger(TMR1, TMR_OVERFLOW_SWTRIG);
```

### 5.18.47 tmr\_output\_enable function

The table below describes the function tmr\_output\_enable

**Table 459. tmr\_output\_enable function**

| Name                 | Description                                                                   |
|----------------------|-------------------------------------------------------------------------------|
| Function name        | tmr_output_enable                                                             |
| Function prototype   | void tmr_output_enable(tmr_type *tmr_x, confirm_state new_state);             |
| Function description | Enable or disable TMR output                                                  |
| Input parameter 1    | tmr_x: indicates the selected TMR peripheral. It can be TMR1, TMR15, TMR16 or |

| Name                   | Description                                                                             |
|------------------------|-----------------------------------------------------------------------------------------|
|                        | TMR17.                                                                                  |
| Input parameter 2      | new_state: TMR output status<br>This parameter can be Enable (TRUE) or Disable (FALSE). |
| Output parameter       | NA                                                                                      |
| Return value           | NA                                                                                      |
| Required preconditions | NA                                                                                      |
| Called functions       | NA                                                                                      |

**Example:**

```
tmr_output_enable(TMR1, TRUE);
```

### 5.18.48 tmr\_internal\_clock\_set function

The table below describes the function tmr\_internal\_clock\_set.

**Table 460. tmr\_internal\_clock\_set function**

| Name                   | Description                                                                        |
|------------------------|------------------------------------------------------------------------------------|
| Function name          | tmr_internal_clock_set                                                             |
| Function prototype     | void tmr_internal_clock_set(tmr_type *tmr_x);                                      |
| Function description   | Set TMR internal clock                                                             |
| Input parameter        | tmr_x: indicates the selected TMR peripheral. It can be TMR1, TMR2, TMR3 or TMR15. |
| Output parameter       | NA                                                                                 |
| Return value           | NA                                                                                 |
| Required preconditions | NA                                                                                 |
| Called functions       | NA                                                                                 |

**Example:**

```
tmr_internal_clock_set(TMR1);
```

## 5.18.49 tmr\_output\_channel\_polarity\_set function

The table below describes the function tmr\_output\_channel\_polarity\_set.

**Table 461. 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, TMR13, TMR14, TMR15, TMR16 or TMR17.                    |
| Input parameter 2      | tmr_channel: Timer channel                                                                                                        |
| 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: | 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         |

### 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);
```

## 5.18.50 tmr\_external\_clock\_config function

The table below describes the function tmr\_external\_clock\_config.

**Table 462. 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 or TMR3.                                                                                     |
| 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: High or rising 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);
```

## 5.18.51 tmr\_external\_clock\_mode1\_config function

The table below describes the function tmr\_external\_clock\_mode1\_config.

Table 463. 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,<br>tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type<br>es_polarity, uint16_t es_filter); |
| Function description   | Configure TMR external clock mode 1 (corresponding to external mode A in the<br>reference manual)                                                                           |
| Input parameter 1      | tmr_x: indicates the selected TMR peripheral. It can be TMR1, TMR2 or TMR3.                                                                                                 |
| 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);
```

## 5.18.52 tmr\_external\_clock\_mode2\_config function

The table below describes the function tmr\_external\_clock\_mode2\_config.

Table 464. 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,<br>tmr_external_signal_divider_type es_divide, tmr_external_signal_polarity_type<br>es_polarity, uint16_t es_filter); |
| Function description | Configure TMR external clock mode 2 (corresponding to external mode B in the<br>reference manual)                                                                           |
| Input parameter 1    | tmr_x : indicates the selected TMR peripheral. It can be TMR1, TMR2 or TMR3.                                                                                                |
| 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);
```

## 5.18.53 tmr\_encoder\_mode\_config function

The table below describes the function tmr\_encoder\_mode\_config.

**Table 465. 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 or TMR3.                                                                                    |
| 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);
```

## 5.18.54 tmr\_force\_output\_set function

The table below describes the function tmr\_force\_output\_set.

Table 466. 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, TMR13, TMR14, TMR15, TMR16 or TMR17.       |
| Input parameter 2      | tmr_channel: timer channel                                                                                           |
| 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);
```

## 5.18.55 tmr\_dma\_control\_config function

The table below describes the function tmr\_dma\_control\_config.

**Table 467. 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, TMR15, TMR16 or TMR17.                              |
| Input parameter 2      | dma_length: DMA transfer length                                                                                               |
| Input parameter 3      | dma_base_address: DMA transfer offset address                                                                                 |
| Output parameter       | NA                                                                                                                            |
| 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);
```

**5.18.56 tmr\_brkdt\_config function**

The table below describes the function tmr\_brkdt\_config.

**Table 468. 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, TMR15, TMR16 or TMR17. |
| 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 at32f425\_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 FCSOEN

**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);
```

## 5.18.57 tmr\_brk\_filter\_value\_get function

The table below describes the function tmr\_brk\_filter\_value\_set.

**Table 469. tmr\_brk\_filter\_value\_set function**

| Name                   | Description                                                                          |
|------------------------|--------------------------------------------------------------------------------------|
| Function name          | tmr_brk_filter_value_set                                                             |
| Function prototype     | void tmr_brk_filter_value_set(tmr_type *tmr_x, uint8_t filter_value)                 |
| Function description   | Set TMR break input filter value                                                     |
| Input parameter 1      | tmr_x: indicates the selected TMR peripheral. It can be TMR1, TMR15, TMR16 or TMR17. |
| Input parameter 2      | filter_value: filter value (from 0x0 to 0xf)                                         |
| Output parameter       | NA                                                                                   |
| Return value           | NA                                                                                   |
| Required preconditions | NA                                                                                   |
| Called functions       | NA                                                                                   |

### Example

filter\_value: filter value (0x0~0xf)

## 5.18.58 tmr\_iremap\_config function

The table below describes the function tmr\_iremap\_config.

**Table 470. tmr\_iremap\_config function**

| Name                   | Description                                                                |
|------------------------|----------------------------------------------------------------------------|
| Function name          | tmr_iremap_config                                                          |
| Function prototype     | void tmr_iremap_config(tmr_type *tmr_x, tmr_input_remap_type input_remap); |
| Function description   | Set TMR internal remapping                                                 |
| Input parameter 1      | tmr_x: indicates the selected TMR peripheral. It can be TMR14.             |
| Input parameter 2      | input_remap: TMR input channel remap to be configured                      |
| Output parameter       | NA                                                                         |
| Return value           | NA                                                                         |
| Required preconditions | NA                                                                         |
| Called functions       | NA                                                                         |

### input\_remap

Set TMR14 channel 1 input remapping

TMR14\_GPIO: TMR14 channel 1 is connected to GPIO

TMR14\_ERTCCLK: TMR14 channel 1 is connected to ERTC clock

TMR14\_HEXT\_DIV32: TMR14 channel 1 is connected to HEXT/32

TMR14\_CLKOUT: TMR14 channel 1 is connected to CLKOUT

### Example

```
tmr_iremap_config(TMR14, TMR14_LICK);
```

## 5.19 Universal synchronous/asynchronous receiver/transmitter (USART)

The USART register structure usart\_type is defined in the “at32f425\_usart.h”:

```
/*
 * @brief type define usart register all
 */
typedef struct
{
 ...
} usart_type;
```

The table below gives a list of the USART registers

**Table 471. 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 472. 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                                  |
| usart_enable                  | Enable USART peripherals                          |
| usart_transmitter_enable      | Enable USART transmitter                          |
| usart_receiver_enable         | Enable USART receiver                             |
| usart_clock_config            | Set clock polarity and phases for synchronization |
| usart_clock_enable            | Set clock output for synchronization              |
| usart_interrupt_enable        | Enable interrupts                                 |
| usart_dma_transmitter_enable  | Enable DMA transmitter                            |
| usart_dma_receiver_enable     | Enable DMA receiver                               |
| usart_wakeup_id_set           | Set wakeup ID                                     |
| usart_wakeup_mode_set         | Set wakeup mode                                   |
| usart_receiver_mute_enable    | Enable receiver mute mode                         |
| usart_break_bit_num_set       | Set break frame length                            |
| usart_lin_mode_enable         | Enable LIN mode                                   |
| usart_data_transmit           | Data transmit                                     |
| usart_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_interrupt_flag_get            | Get interrupt flag status                                                  |
| uart_flag_clear                    | Clear flag                                                                 |
| uart_rs485_delay_time_config       | Set latency for starting or ending consecutive data transmission in RS 485 |
| uart_transmit_receive_pin_swap     | Swap transmit/receive pins                                                 |
| uart_id_bit_num_set                | Set ID bit count                                                           |
| uart_de_polarity_set               | DE signal polarity selection                                               |
| uart_rs485_mode_enable             | RS485 mode enable                                                          |

### 5.19.1 usart\_reset function

The table below describes the function usart\_reset.

**Table 473. 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);
```

## 5.19.2 usart\_init function

The table below describes the function usart\_init.

Table 474. 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 bits.                                                                                       |
| 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 bits

USART\_STOP\_1\_5\_BIT: 1.5 bits

### Example:

```
/* configure uart param */
usart_init(USART1, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
```

### 5.19.3 usart\_parity\_selection\_config function

The table below describes the function usart\_parity\_selection\_config.

**Table 475. usart\_parity\_selection\_config function**

| Name                   | Description                                                                                     |
|------------------------|-------------------------------------------------------------------------------------------------|
| Function name          | usart_parity_selection_config                                                                   |
| Function prototype     | void usart_parity_selection_config(usart_type* usart_x,<br>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);
```

### 5.19.4 usart\_enable function

The table below describes the function usart\_enable.

**Table 476. 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);
```

## 5.19.5 usart\_transmitter\_enable function

The table below describes the function usart\_transmitter\_enable.

**Table 477. 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);
```

## 5.19.6 usart\_receiver\_enable function

The table below describes the function usart\_receiver\_enable.

**Table 478. 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);
```

## 5.19.7 usart\_clock\_config function

The table below describes the function usart\_clock\_config.

**Table 479. 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:<br>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);
```

## 5.19.8 usart\_clock\_enable function

The table below describes the function usart\_clock\_enable.

**Table 480. 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);
```

## 5.19.9 usart\_interrupt\_enable function

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

**Table 481. 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 interrupt
- USART\_CTSCF\_INT: CTS (Clear To Send) change

**Example:**

```
/* enable usart1 transmit complete interrupt */
usart_interrupt_enable (USART1, USART_TDC_INT, TRUE);
```

**5.19.10 usart\_dma\_transmitter\_enable function**

The table below describes the function usart\_dma\_transmitter\_enable.

**Table 482. 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);
```

**5.19.11 usart\_dma\_receiver\_enable function**

The table below describes the function usart\_dma\_receiver\_enable.

**Table 483. 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);
```

## 5.19.12 usart\_wakeup\_id\_set function

The table below describes the function usart\_wakeup\_id\_set.

**Table 484. 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);
```

## 5.19.13 usart\_wakeup\_mode\_set function

The table below describes the function usart\_wakeup\_mode\_set.

**Table 485. 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);
```

### 5.19.14 usart\_receiver\_mute\_enable function

The table below describes the function usart\_receiver\_mute\_enable.

**Table 486. 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);
```

### 5.19.15 usart\_break\_bit\_num\_set function

The table below describes the function usart\_break\_bit\_num\_set.

**Table 487. 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);
```

### 5.19.16 usart\_lin\_mode\_enable function

The table below describes the function usart\_lin\_mode\_enable.

**Table 488. 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);
```

### 5.19.17 usart\_data\_transmit function

The table below describes the function usart\_data\_transmit.

**Table 489. 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);
```

### 5.19.18 usart\_data\_receive function

The table below describes the function usart\_data\_receive.

**Table 490. 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);
```

### 5.19.19 usart\_break\_send function

The table below describes the function usart\_break\_send.

**Table 491. 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);
```

## 5.19.20 usart\_smartcard\_guard\_time\_set function

The table below describes the function usart\_smartcard\_guard\_time\_set.

**Table 492. 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);
```

## 5.19.21 usart\_irda\_smartcard\_division\_set function

The table below describes the function usart\_irda\_smartcard\_division\_set.

**Table 493. 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);
```

## 5.19.22 usart\_smartcard\_mode\_enable function

The table below describes the function usart\_smartcard\_mode\_enable.

**Table 494. 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);
```

## 5.19.23 usart\_smartcard\_nack\_set function

The table below describes the function usart\_smartcard\_nack\_set.

**Table 495. 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);
```

## 5.19.24 usart\_single\_line\_halfduplex\_select function

The table below describes the function usart\_single\_line\_halfduplex\_select.

**Table 496. 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);
```

## 5.19.25 usart\_irda\_mode\_enable function

The table below describes the function usart\_irda\_mode\_enable.

**Table 497. 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);
```

## 5.19.26 usart\_irda\_low\_power\_enable function

The table below describes the function usart\_irda\_low\_power\_enable.

**Table 498. 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);
```

## 5.19.27 usart\_hardware\_flow\_control\_set function

The table below describes the function usart\_hardware\_flow\_control\_set.

**Table 499. 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 hardware 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);
```

## 5.19.28 usart\_flag\_get function

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

Table 500. 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                                                                       |
| 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);
```

## 5.19.29 usart\_interrupt\_flag\_get function

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

Table 501. 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   | Get USART interrupt flag status                                           |
| Input parameter 1      | usart_x: indicates the selected peripheral. It can be USART1 or USART2.   |
| Input parameter 2      | Flag: clear the selected flag                                             |
| Output parameter       | NA.                                                                       |
| Return value           | flag_status: SET or RESET                                                 |
| Required preconditions | NA                                                                        |
| Called functions       | NA                                                                        |

### flag

|                  |                                 |
|------------------|---------------------------------|
| USART_CTSFLAG:   | CTS (Clear To Send) change flag |
| USART_BFFFLAG:   | Break frame receive flag        |
| USART_TDBEFLAG:  | Transmit buffer empty flag      |
| USART_TDCFLAG:   | Transmit complete flag          |
| USART_RDBFFLAG:  | Receive data buffer full flag   |
| USART_IDLEFLAG:  | Idle frame flag                 |
| USART_ROERRFLAG: | Receive overflow flag           |
| USART_NERRFLAG:  | Noise error flag                |
| USART_FERRFLAG:  | Frame error flag                |
| USART_PERRFLAG:  | Parity error flag               |

### Example:

```
/* check received data flag */
if(usart_interrupt_flag_get(USART1, USART_RDBFFLAG) != RESET)
{
}
```

### 5.19.30 usart\_flag\_clear function

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

**Table 502. 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_CTS_CF_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);
```

### 5.19.31 usart\_rs485\_delay\_time\_config function

The table below describes the function usart\_rs485\_delay\_time\_config.

**Table 503. usart\_flag\_clear function**

| Name                   | Description                                                                                                                                                                                                |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Function name          | usart_rs485_delay_time_config                                                                                                                                                                              |
| Function prototype     | void usart_rs485_delay_time_config(usart_type* usart_x, uint8_t start_delay_time, uint8_t complete_delay_time);                                                                                            |
| Function description   | Delay time for setting and clearing data valid signal for continuous data transmission in RS485 mode                                                                                                       |
| Input parameter 1      | usart_x: indicates the selected peripheral. It can be USART1, USART2 and USART3.                                                                                                                           |
| Input parameter 2      | start_delay_time:<br>After the first data is written in continuous transmit mode, data valid signal is set and “start_delay_time” is inserted before sending data. The time unit is 1/16 baud rate period. |
| Input parameter 3      | complete_delay_time:<br>After the last data is sent in continuous transmit mode, the “complete_delay_time” is inserted before clearing data valid signal. The time unit is 1/16 baud rate period.          |
| Output parameter       | NA                                                                                                                                                                                                         |
| Return value           | NA                                                                                                                                                                                                         |
| Required preconditions | NA                                                                                                                                                                                                         |
| Called functions       | NA                                                                                                                                                                                                         |

**Example:**

```
/* config rs485 delay time */
usart_rs485_delay_time_config(USART1, 2, 2);
```

### 5.19.32 usart\_transmit\_receive\_pin\_swap function

The table below describes the function usart\_transmit\_receive\_pin\_swap.

**Table 504. usart\_transmit\_receive\_pin\_swap function**

| Name                   | Description                                                                         |
|------------------------|-------------------------------------------------------------------------------------|
| Function name          | usart_transmit_receive_pin_swap                                                     |
| Function prototype     | void usart_transmit_receive_pin_swap(usart_type* usart_x, confirm_state new_state); |
| Function description   | Swap transmit/receive pins                                                          |
| Input parameter 1      | usart_x: indicates the selected peripheral. It can be USART1, USART2 and USART3.    |
| Input parameter 2      | new_state: enabled (TRUE) or disabled (FALSE)                                       |
| Output parameter       | NA                                                                                  |
| Return value           | NA                                                                                  |
| Required preconditions | NA                                                                                  |
| Called functions       | NA                                                                                  |

**Example:**

```
/* enable tx/rx swap */
usart_transmit_receive_pin_swap (USART1, TRUE);
```

### 5.19.33 usart\_id\_bit\_num\_set function

The table below describes the function usart\_id\_bit\_num\_set.

**Table 505. usart\_transmit\_receive\_pin\_swap function**

| Name                   | Description                                                                                   |
|------------------------|-----------------------------------------------------------------------------------------------|
| Function name          | usart_id_bit_num_set                                                                          |
| Function prototype     | void usart_id_bit_num_set(usart_type* usart_x, usart_identification_bit_num_type id_bit_num); |
| Function description   | Set ID bit number                                                                             |
| Input parameter 1      | usart_x: indicates the selected peripheral. It can be USART1, USART2 and USART3.              |
| Input parameter 2      | id_bit_num: ID bit count                                                                      |
| Output parameter       | NA                                                                                            |
| Return value           | NA                                                                                            |
| Required preconditions | NA                                                                                            |
| Called functions       | NA                                                                                            |

**id\_bit\_num**

|                            |                     |
|----------------------------|---------------------|
| USART_ID_FIXED_4_BIT:      | 4-bit ID            |
| USART_ID RELATED DATA BIT: | Current data bit -1 |

**Example:**

```
/* config ID bit width */
usart_id_bit_num_set (USART1, USART_ID_FIXED_4_BIT);
```

### 5.19.34 usart\_de\_polarity\_set function

The table below describes the function usart\_de\_polarity\_set.

**Table 506. usart\_de\_polarity\_set function**

| Name                   | Description                                                                          |
|------------------------|--------------------------------------------------------------------------------------|
| Function name          | usart_de_polarity_set                                                                |
| Function prototype     | void usart_de_polarity_set(usart_type* usart_x, usart_de_polarity_type de_polarity); |
| Function description   | DE signal polarity selection                                                         |
| Input parameter 1      | usart_x: indicates the selected peripheral. It can be USART1, USART2 and USART3.     |
| Input parameter 2      | de_polarity: DE signal polarity                                                      |
| Output parameter       | NA                                                                                   |
| Return value           | NA                                                                                   |
| Required preconditions | NA                                                                                   |
| Called functions       | NA                                                                                   |

#### de\_polarity

USART\_DE\_POLARITY\_HIGH: DE signal active high

USART\_DE\_POLARITY\_LOW: DE signal active low

#### Example:

```
/* config DE polarity */
usart_de_polarity_set(USART1, USART_DE_POLARITY_HIGH);
```

### 5.19.35 usart\_rs485\_mode\_enable function

The table below describes the function usart\_rs485\_mode\_enable.

**Table 507. usart\_rs485\_mode\_enable function**

| Name                   | Description                                                                      |
|------------------------|----------------------------------------------------------------------------------|
| Function name          | usart_rs485_mode_enable                                                          |
| Function prototype     | void usart_rs485_mode_enable(usart_type* usart_x, confirm_state new_state);      |
| Function description   | RS485 mode enable                                                                |
| Input parameter 1      | usart_x: indicates the selected peripheral. It can be USART1, USART2 and USART3. |
| Input parameter 2      | new_state: enabled (TRUE) or disabled (FALSE)                                    |
| Output parameter       | NA                                                                               |
| Return value           | NA                                                                               |
| Required preconditions | NA                                                                               |
| Called functions       | NA                                                                               |

#### Example:

```
/* enable rs485 mode */
usart_rs485_mode_enable (USART1, TRUE);
```

## 5.20 Watchdog timer (WDT)

The WDT register structure `wdt_type` is defined in the “`at32f425_wdt.h`”:

```
/**
 * @brief type define wdt register all
 */

typedef struct
{

} wdt_type;
```

The table below gives a list of the WDT registers

**Table 508. Summary of WDT registers**

| Register | Description      |
|----------|------------------|
| cmd      | Command register |
| div      | Divider register |
| rld      | Reload register  |
| sts      | Status register  |
| win      | Window register  |

The table below gives a list of WDT library functions.

**Table 509. Summary of WDT library functions**

| Function name                          | Description                                          |
|----------------------------------------|------------------------------------------------------|
| <code>wdt_enable</code>                | Enable watchdog                                      |
| <code>wdt_counter_reload</code>        | Reload counter                                       |
| <code>wdt_reload_value_set</code>      | Set reload value                                     |
| <code>wdt_divider_set</code>           | Set division value                                   |
| <code>wdt_register_write_enable</code> | Unlock WDT_DIV and WDT_RLD register write protection |
| <code>wdt_flag_get</code>              | Get flag                                             |
| <code>wdt_window_counter_set</code>    | Set window counter                                   |

## 5.20.1 wdt\_enable function

The table below describes the function wdt\_enable.

**Table 510. 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();
```

## 5.20.2 wdt\_counter\_reload function

The table below describes the function wdt\_counter\_reload.

**Table 511. 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();
```

### 5.20.3 `wdt_reload_value_set` function

The table below describes the function `wdt_reload_value_set`.

**Table 512. `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);
```

### 5.20.4 `wdt_divider_set` function

The table below describes the function `wdt_divider_set`.

**Table 513. `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<br>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);
```

## 5.20.5 wdt\_register\_write\_enable function

The table below describes the function wdt\_register\_write\_enable.

**Table 514. 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<br>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);
```

## 5.20.6 wdt\_flag\_get function

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

**Table 515. 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<br>Refer to the "flag" description below for details. |
| Output parameter       | NA                                                                         |
| Return value           | flag_status: flag status<br>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

WDT\_WINF\_UPDATE\_FLAG: Window value update complete

**Example:**

```
wdt_flag_get(WDT_DIVF_UPDATE_FLAG);
```

## 5.20.7 `wdt_window_counter_set` function

The table below describes the function `wdt_window_counter_set`.

**Table 516. `wdt_window_counter_set` function**

| Name                   | Description                                                  |
|------------------------|--------------------------------------------------------------|
| Function name          | <code>wdt_window_counter_set</code>                          |
| Function prototype     | <code>void watchdog_counter_set(uint16_t window_cnt);</code> |
| Function description   | Set window counter                                           |
| Input parameter        | <code>window_cnt</code> : window value, from 0x000 to 0xFFFF |
| Output parameter       | NA                                                           |
| Return value           | NA                                                           |
| Required preconditions | NA                                                           |
| Called functions       | NA                                                           |

**Example:**

```
watchdog_counter_set(0x7FF);
```

## 5.21 Window watchdog timer (WWDT)

The WWDT register structure wwdt\_type is defined in the “at32f425\_wwdt.h”:

```
/**
 * @brief type define wwdt register all
 */

typedef struct
{

} wwdt_type;
```

The table below gives a list of the WWDT registers

**Table 517. 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 518. 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                    |

## 5.21.1 wwdt\_reset function

The table below describes the function wwdt\_reset.

**Table 519. 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();
```

## 5.21.2 wwdt\_divider\_set function

The table below describes the function wwdt\_divider\_set.

**Table 520. 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<br>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);
```

### 5.21.3 wwdt\_enable function

The table below describes the function wwdt\_enable.

**Table 521. 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);
```

### 5.21.4 wwdt\_interrupt\_enable function

The table below 3 describes the function wwdt\_interrupt\_enable.

**Table 522. 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();
```

### 5.21.5 wwdt\_counter\_set function

The table below describes the function wwdt\_counter\_set.

**Table 523. 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);
```

## 5.21.6 wwdt\_window\_counter\_set function

The table below describes the function wwdt\_window\_counter\_set.

**Table 524. 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 counter 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);
```

## 5.21.7 wwdt\_flag\_get function

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

**Table 525. 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<br>Return SET or RESET |
| Required preconditions | NA                                              |
| Called functions       | NA                                              |

**Example:**

```
wwdt_flag_get();
```

## 5.21.8 wwdt\_interrupt\_flag\_get function

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

**Table 526. wwdt\_interrupt\_flag\_get 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 status, and check corresponding interrupt enable bit. |
| Input parameter        | NA                                                                                      |
| Output parameter       | NA                                                                                      |
| Return value           | flag_status: SET or RESET                                                               |
| Required preconditions | NA                                                                                      |
| Called functions       | NA                                                                                      |

**Example:**

```
wwdt_interrupt_flag_get();
```

## 5.21.9 wwdt\_flag\_clear function

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

**Table 527. 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();
```

## 6 Precautions

### 6.1 Device model replacement

While replacing the device part number in an existing project or demo with another one, if necessary, it is necessary to check the macro definitions corresponding to the device defined in [Table 1](#) before replacement. The subsequent sections give a detailed description of how to replace a device in KEIL and IAR environments (Just taking the at32f403avgt7 as an example as other devices share similar operations).

There are two steps to get this happen:

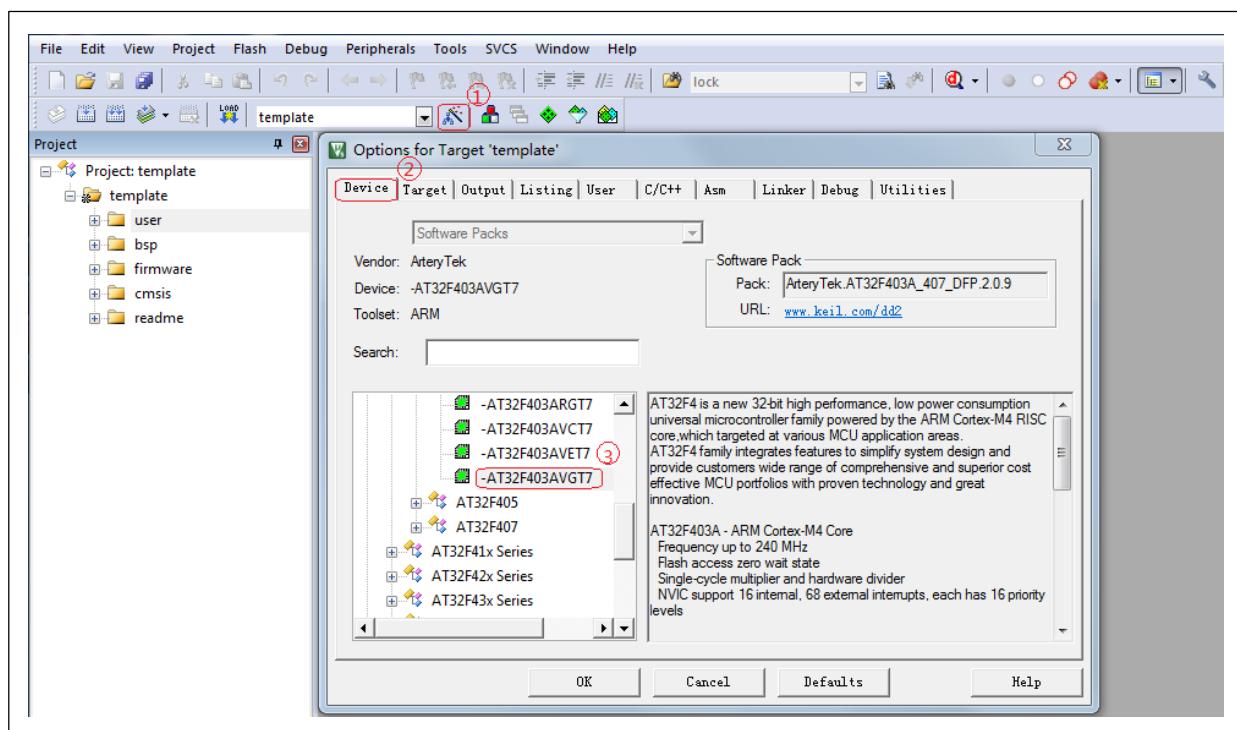
1. By changing device
2. By changing macro definition

#### 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

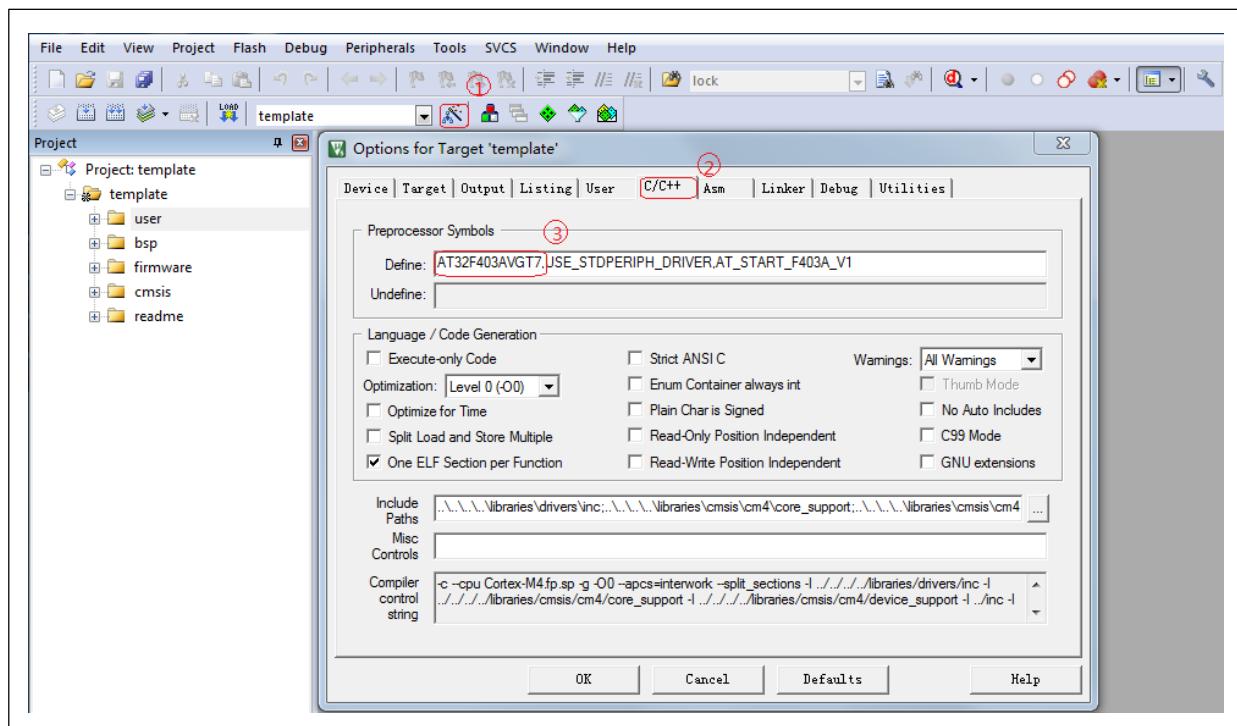
**Figure 29. Change device part number in Keil**



Follow the steps and illustration below to change 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 one corresponding to the selected device part number based o [Table 1](#).

Figure 30. Change macro definition in Keil

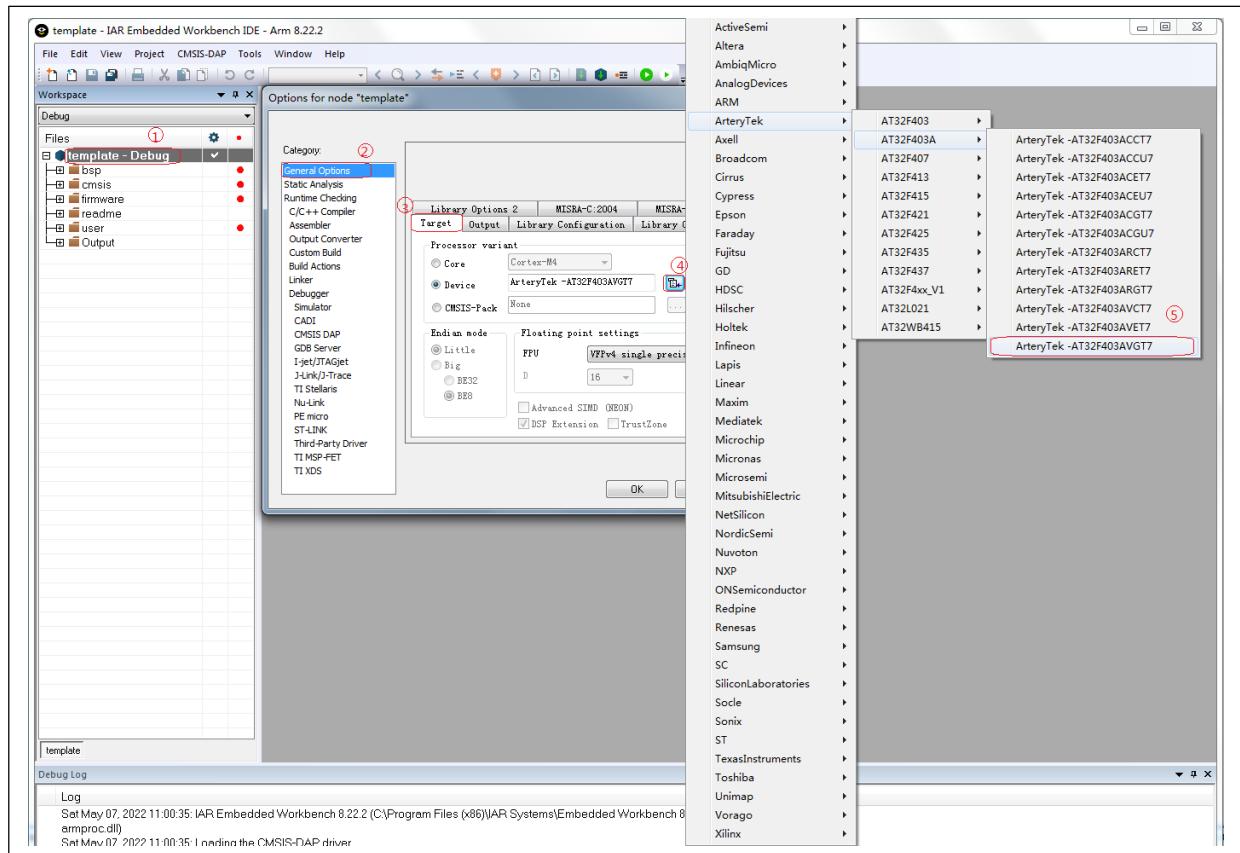


## 6.1.2 IAR environment

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.

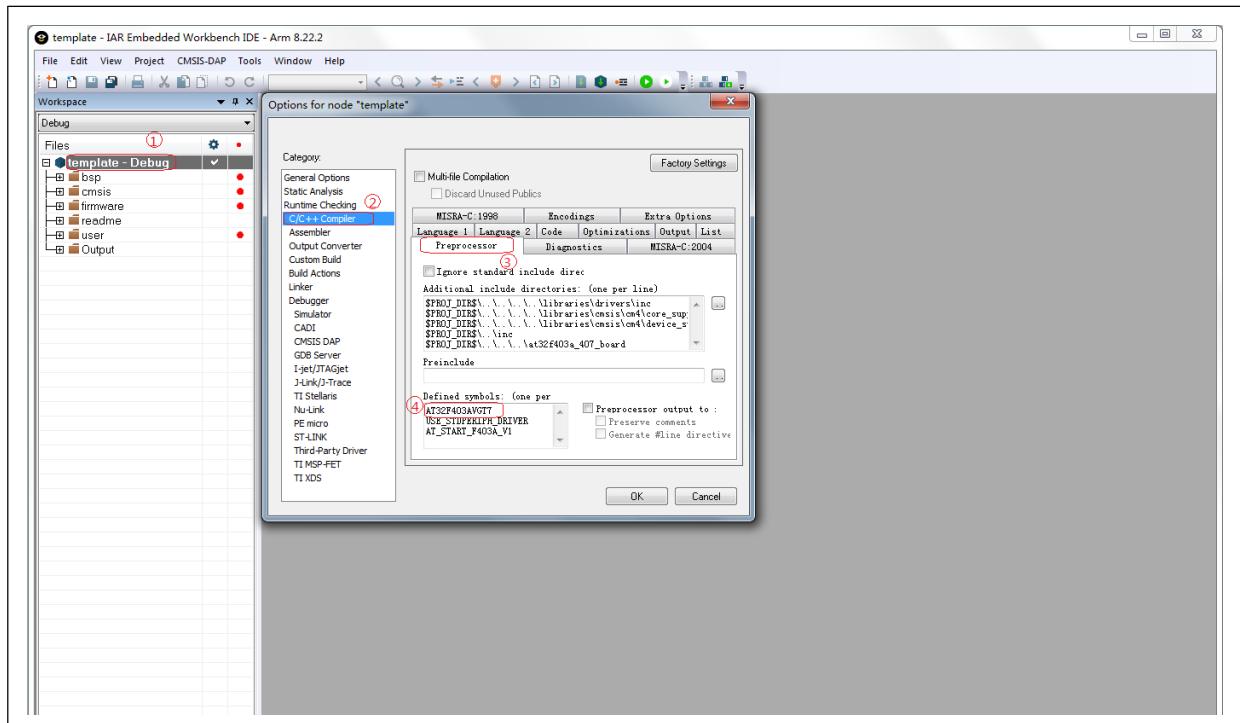
Figure 31. Change device part number in IAR



Follow the steps and illustration below to change 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 one corresponding to the selected device part number based on [Table 1](#).

Figure 32. Change macro definition in IAR



## 6.2 Unable 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

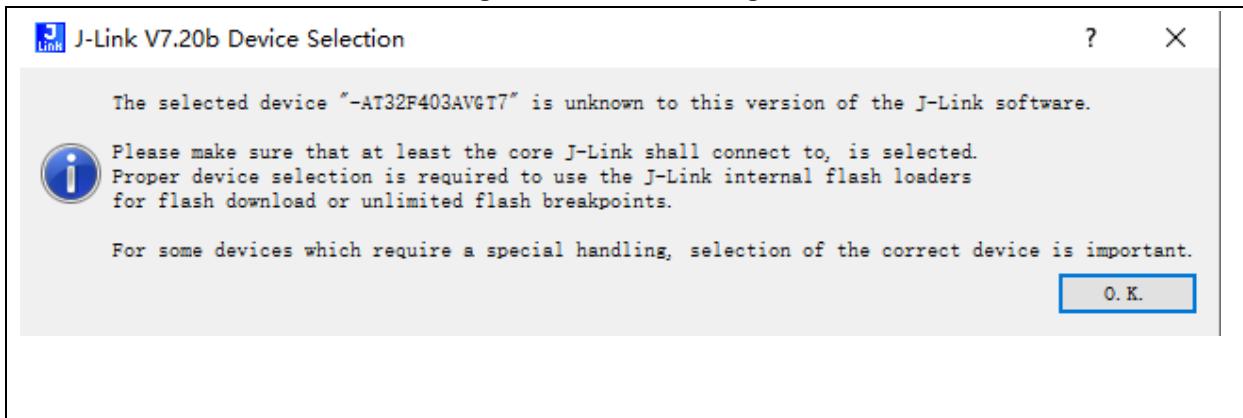


Figure 34. Error warning 2

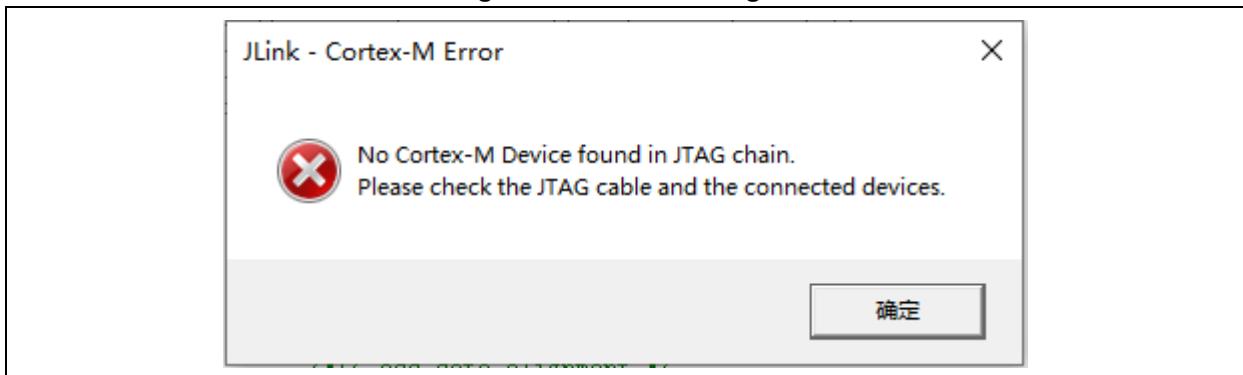
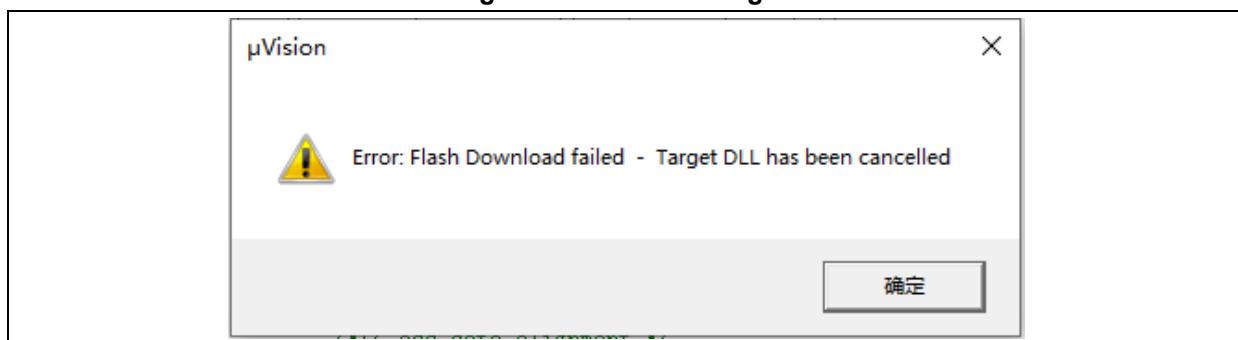


Figure 35. Error warning 3

**How to solve this problem?**

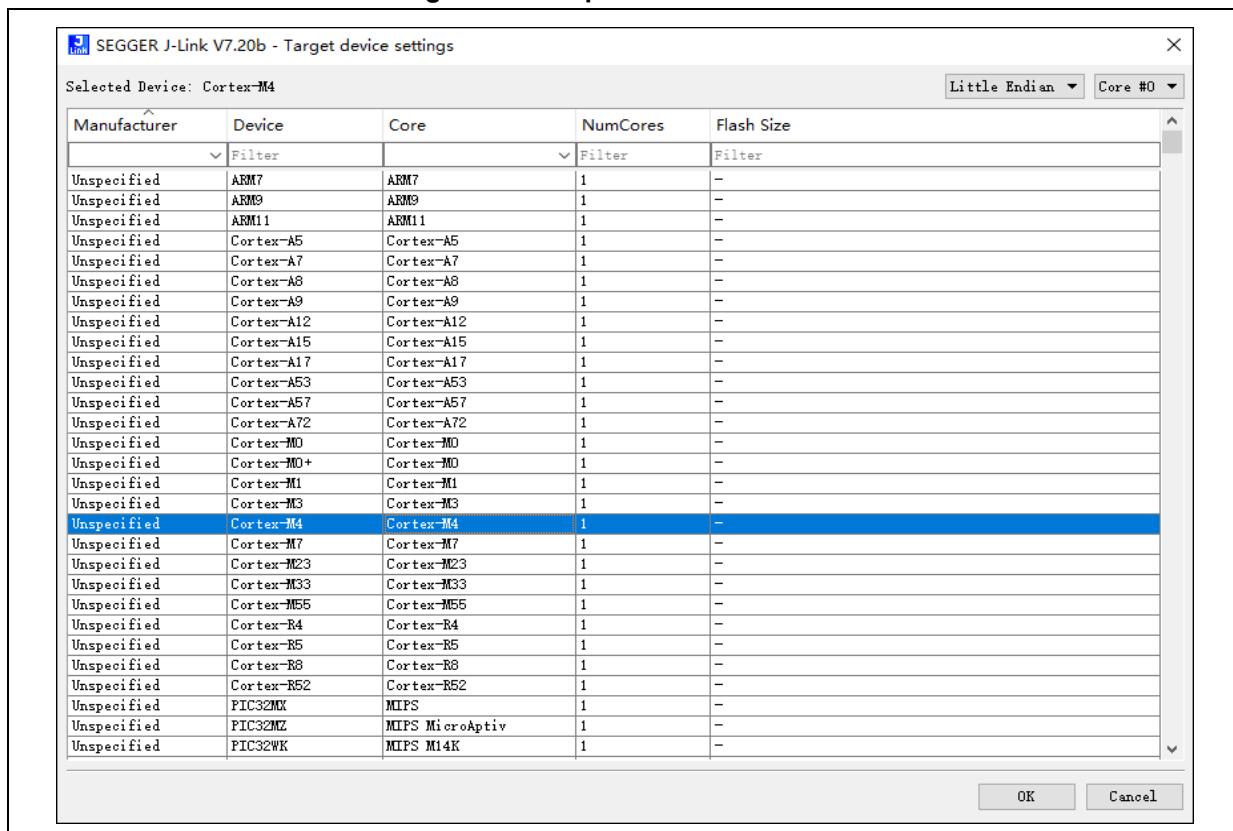
Step 1: Find “JLinkLog” and “JLinkSettings” files according to project path, and delete them.

Figure 36. JLinkLog and JLinkSettings

| 名称                                  | 修改日期            | 类型                | 大小    |
|-------------------------------------|-----------------|-------------------|-------|
| listings                            | 2022/2/22 19:28 | 文件夹               |       |
| objects                             | 2022/2/22 19:28 | 文件夹               |       |
| combine_mode_ordinary_simult.uvoptx | 2022/2/22 19:28 | UVOPTX 文件         | 12 KB |
| combine_mode_ordinary_simult        | 2022/2/22 19:28 | Majision5 Project | 17 KB |
| JLinkLog                            | 2022/2/22 19:28 | 文本文档              | 7 KB  |
| JLinkSettings                       | 2022/2/22 19:27 | 配置设置              | 1 KB  |

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.

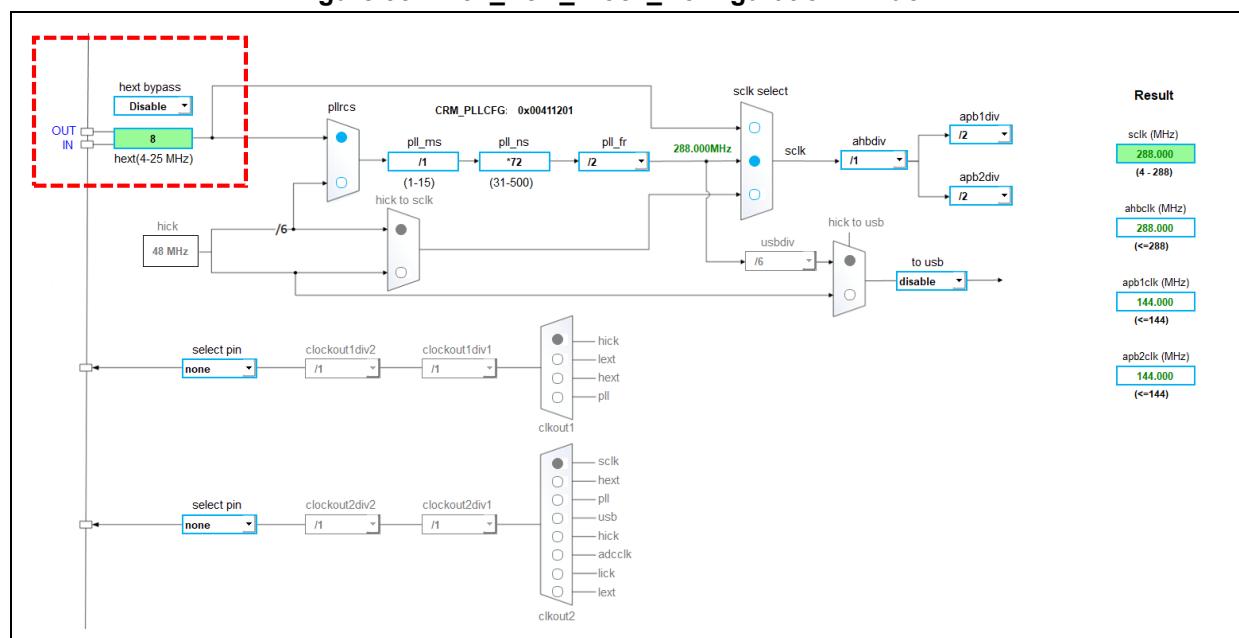
The users simply need to replace the original one in BSP demo with the newly generated clock code file (at32f4xx\_clock.c/ at32f4xx\_clock.h/ at32f4xx\_conf.h), and call the function system\_clock\_config in main function.

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 589, which are all available from the official website of Artery.

**Table 528. 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 529. Document revision history

| Date       | Revision | Changes                                                                                                   |
|------------|----------|-----------------------------------------------------------------------------------------------------------|
| 2021.11.12 | 2.0.0    | Initial release                                                                                           |
| 2022.06.15 | 2.0.1    | Added <a href="#">section 5 AT32F425 peripheral library functions</a>                                     |
| 2022.11.15 | 2.0.2    | Modified the I2C abbreviations in the <a href="#">section 4.2.1 List of abbreviations for peripherals</a> |
| 2023.07.18 | 2.0.3    | Added <a href="#">section 5.4.10 to 5.4.13</a>                                                            |
| 2023.10.26 | 2.0.4    | Added the function “interrupt_flag_get” to each section of this file.                                     |
| 2024.03.01 | 2.0.5    | Changed “PWC” to “NVIC” in the titles of Table 354 and Table 355 of <a href="#">Section 5.13</a>          |

**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 granted 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 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 on 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 characteristics stated in this document shall immediately void any warranty grant by ARTERY for ARTERY's products or services described herein and shall not create or expand any liability of ARTERY in any manner whatsoever.